Mint™ version 4

Advanced Programming Guide

MN1270

MN1270 02.2002

Baldor UK Ltd
Mint Motion Centre
6 Bristol Distribution Park
Hawkley Drive
Bristol
BS32 0BF
U.K.

Telephone:	+44 (0) 1454 850 000
Fax:	+44 (0) 1454 859 001
Web site:	www.baldor.co.uk
Sales email:	sales@baldor.co.uk
Support email:	technical.support@baldor.co.uk

Baldor Electric Company

Telephone:	+1 501 646 4711
Fax:	+1 501 648 5792
email:	sales@baldor.com
web site:	www.baldor.com

Baldor ASR GmbH

Telephone:	+49 (0) 89 90508-0
Fax:	+49 (0) 89 90508-492

Baldor ASR AG

Telephone:	+41 (0) 52 647 4700
Fax:	+41 (0) 52 659 2394

Australian Baldor Pty Ltd

Telephone:	+61 2 9674 5455
Fax:	+61 2 9674 2495

Baldor Electric (F.E.) Pte Ltd

Telephone:	+65 744 2572

Fax:+65 747 1708

SAFETY NOTICE:

Only qualified personnel should attempt the start-up procedure or troubleshoot this equipment. This equipment may be connected to other machines that have rotating parts or parts that are controlled by this equipment. Improper use can cause serious or fatal injury. Only qualified personnel should attempt to start-up, program or troubleshoot this equipment.

Precautions:

WARNING: Be sure that you are completely familiar with the safe operation of this equipment. This equipment may be connected to other machines that have rotating parts or parts that are controlled by this equipment. Improper use can cause serious or fatal injury. Only qualified personnel should attempt to program, start-up or troubleshoot this equipment.

WARNING: Be sure that you are completely familiar with the safe programming of this equipment. This equipment may be connected to other machines that have rotating parts or parts that are controlled by this equipment. Improper programming of this equipment can cause serious or fatal injury. Only qualified personnel should attempt to program, start-up or troubleshoot this equipment.

WARNING: The stop input to this equipment should not be used as the single means of achieving a safety critical stop. Drive disable, motor disconnect, motor brake and other means should be used as appropriate. Only qualified personnel should attempt to program, start-up or troubleshoot this equipment.

WARNING: Improper operation or programming of the control may cause violent motion of the motor shaft and driven equipment. Be certain that unexpected motor shaft movement will not cause injury to personnel or damage to equipment. Peak torque of several times the rated motor torque can occur during control failure.

WARNING: The motor shaft will rotate during the homing procedure. Be certain that unexpected motor shaft movement will not cause injury to personnel or damage to equipment.

CAUTION: To prevent equipment damage, be certain that input and output signals are powered and referenced correctly.

Manual Revision History

Issue	Date	BOCL Reference	Comments
1.0 Draft A	Apr '99	UM00546-000	First draft.
1.1	July '99	UM00546-001	Made corrections.
1.2	January 2000	UM00546-002	Further corrections.
1.3	August 2000	UM00546-003	Updates for Mint v4.2
1.4	April 2001	UM00546-004	Updates from 04.2001 Errata
1.5	February 2002	UM00546-005	Updates from 11.2001 Errata

Introduction

1

Mint™ is a flexible Basic like programming language designed for high speed motion control applications.

This Mint v4 Advanced Programming Guide accompanies the Mint v4 Programming Guide. It covers the advanced Mint features:

- Spline moves on NextMove
- Hold to Analog whereby the position is held on the basis of an analog feedback signal.
- DAC monitoring, allows a DAC output to output values such as axis velocity or following error.
- Data capture. This feature is used by the Mint WorkBench. This manual describes the keywords used for data capture.
- Current limiting. MintDrive supports I^2t, full details of the keywords are described.
- NextMove supports analog input errors. Analog inputs can be assigned to an axis. If there go out of a programmable range, then an error will be generated.
- Leadscrew compensation allows a leadscrew or ballscrew real positions to be mapped to axis positions.
- Backlash compensation.
- NextMove PC expansion.

This manual is for advanced Mint users only. If you are unfamiliar with Mint then please refer the Mint v4 Programming Guide.

Multi Axis Mappings and Configurations

2

The NextMove family of controllers support multiple axes of control. This chapter covers:

◊ Axes supported by NextMove controllers
◊ Axis and Channel numbering convention
◊ Reconfiguring the axes in Mint

The NextMove family of controllers are multi-axis controllers. Depending on the hardware available these axes may be used to control either servo or stepper axes. The number of axes supported by each controller is detailed below:

Controller	Number of Software Axes	Number of Physical Axes	Servo	Stepper
NextMove BX	8	4	●	
NextMove PC	8	8	●	●
NextMove PCI	12	8[1]	●	●

A servo axis consists of a DAC and encoder channel. A stepper axis consists of a pulse and direction output. A controller may have more axes available in software than the number of axes in hardware. This allows a number of virtual axes to be configured.

A NextMove BX supports up to 4 servo axes and is sold in 2, 3, and 4 axis variants.

A NextMove PC supports 4 servo axes and 4 stepper axes. There is an axis expansion card available for the NextMove PC controller which supports another 4 servos and 4 steppers allowing the controller to be used to control up to 8 axes of servos or 8 axes of steppers.

A NextMove PCI supports 4 servo axes and 4 stepper axes. There is also an axis expansion card available for the NextMove PCI controller which supports another 4 servos and 4 steppers. Up to two expansion cards can be linked to a single main card allowing the controller to be used to control up to 12 axes of servos or 12 axes of steppers. The NextMove PCI main card is sold in 1, 2, 3, 4 and 8 axis variants.

The **`VIEW PLATFORM`** keyword shows the number of axes, the number of axes of control and the number of hardware channels available on the card.

The **`VIEW CONFIG`** keyword shows the configuration of each axis and the hardware channel being used.

The **`CONFIG`** keyword is used to change the configuration of an axis. It is possible to change the hardware channel that an axis is using. See the **`AXISCHANNEL`** keyword for details.

2.1 Axis Variants

Controllers are sold as being able to control a number of axes. An axis of control means that an axis can be configured for closed loop control, i.e. a servo axis or a stepper axis. For example, on a 2 axis controller, it is only possible to have two closed loop axes running. Each axis of control can be configured as a servo or a stepper axis (subject to availability of hardware).

On each card, there are a number of hardware resources. These are called channels and are always numbered sequentially from zero. For example, the first encoder, DAC output and stepper pulse and direction outputs are all referenced as channel zero.

Regardless of the number of axes of control purchased, the number of axes in software is always constant. Any axes that are not controlling actual hardware can be configured to be virtual axes.

NextMove controllers support ‘virtual axes’. A virtual axis allows motion to be simulated without moving any physical axes. To is useful for system design and testing. A virtual axis allows most Mint commands to be executed as normal and the axis will simulate position and velocity information for any motion performed. To configure an axis as a virtual axis, the **`CONFIG`** keyword is used. Any axis can be configured to be a virtual axis. NextMove PCI and NextMove BX have additional software axes that be configured to be virtual axes without taking up any of the regular axes connected to physical hardware.

Any hardware that is not being used for closed loop control is available to the user for open loop control. When addressing hardware for open loop control, the hardware's channel number must be used, NOT the axis number of the axis that was associated to the hardware. It is possible for an axis number not to match the channel number of the hardware it is using.

[1] Using a NextMove PCI expansion card will increase the number of physical axes to a maximum of 12.

Care must be taken to see if a Mint keyword is axis based or channel based to ensure that the correct parameter is passed. For example, assume that axis 4 is configured as a servo axis and is using hardware channel 0. To set the default action in event of a following error to be ramp the DAC to zero, you would call **FOLERRORMODE** on axis 4 as **FOLERRORMODE** is an axis based keyword.

```
FOLERRORMODE.4 = _emDACRAMP
```

To set the DAC ramp rate, the **DACRAMP** keyword is used. This is a channel based keyword so the keyword is called for channel 0.

```
DACRAMP.0 = 100
```

The read which channel an axis using, the **AXISCHANNEL** keyword can be read.

```
FOLERRORMODE.4 = _emDACRAMP
channel = AXISCHANNEL.4
DACRAMP.channel = 100
```

Creating Motion

3

This chapter covers the use of Mint to perform motion. The following areas are included, and instruction provided to allow the user to explore motion control through example programs:

- ◊ Encoder following and gearing
- ◊ Compensation modes
- ◊ Spline profiles

3.1 Gearing Compensation

All sampled master / slave systems have an inherent lag in the system. This lag is speed dependant, the faster the master axis, the larger the lag seen. Gearing compensation can be used to overcome this lag for the **FOLLOW**, **FLY** and **CAM** move types.

The **GEARING** keyword allows the lag or lead position of the slave axis to be controlled or removed. The compensation is accomplished by feeding forward the slave velocity.

Example:

```
MASTERSOURCE.0 = 0    : REM Master is axis position
MASTERCHANNEL.0 = 3   : REM Master is axis 3
GEARING.0 = 100.0     : REM Compensate lag
GEARINGMODE.0 = 1     : REM Turn on gearing compensation
FOLLOW.0 = 1.0        : REM Start following the master
```

3.2 Spline and P.V.T.

Splines are only available on the NextMove product range.

A SPLINE move provides a means of specifying motion where arbitrary position and velocity information needs to be specified in terms of time. The resulting motion is then calculated such that the motion profile is continuous in position and velocity (acceleration and jerk also if no velocity data is given). The Spline motion is defined using a number of segments. Each segment defines the how far the slave axis will travel in a specified time. The Spline segments are placed in a table for a background execution. Once these segments have been defined, NextMove will interpolate between them and 'fill in' the missing information, producing a smooth path as it goes. Splines can be performed on any number of servo or stepper axes.

3.2.1 Spline Tables

A spline cycle is broken up into a series of segments which make up the profile. A spline table is set-up in an array of any name where the normal Mint syntax applies to the array.

For example:

```
DIM myPos(11) = 10,1,2,3,2,1,2,3,1,2,3
```

where the first element determines the number of points in the spline profile and each subsequent value defines a spline segment. A segment can be defined in either relative or absolute positions . The type of spline performed is determined by value passed to the **SPLINE** keyword.

There are three types of profile available:

1. Spline 1. Only position data is used and a smooth path (in position, velocity, acceleration and jerk) is produced. The axis may not pass through the points specified in the spline table, depending on the values.
2. Spline 2. Only position data is used and a smooth path (in position and velocity) is produced. The axis will pass through the points specified in the spline table.
3. PVT. Position and velocity information is used to produce a smooth path (in position and velocity). The velocity information for each segment is defined in array. At the segment, the axis will pass through the points specified at the velocity specified.

Before the spline can be executed, a segment duration must be defined. This is done with the **SPLINETIME** keyword which specifies a duration in milliseconds. Alternatively, individual segment durations can be specified by the use of a duration array.

The **SPLINETABLE** keyword is used tell NextMove which arrays to use for the spline. **SPLINETABLE** is a function style keyword. For example, to use the position array *myPos* and the duration array myDur:

```
SPLINETABLE ( 2, myPos, NULL, myDur )
```

where 2 is the axis number. The **NULL** parameter indicates that there is no velocity data.

The **SPLINE** keyword is used to load the move and the **GO** keyword starts the motion. The value passed to the **SPLINE** keyword indicates the type of spline, whether the spline is to repeat and whether the values in the table are to be taken as relative or absolute. For example:

```
SPLINE.0 = _spSPLINE_1 : GO.0
```

will execute a Spline 1 move for axis 0 taking the positions in the spline table as relative. The spline will finish when the last segment has been executed.

```
SPLINE.1 = _spABSOLUTE + _spT_ABSOLUTE + _spCONTINUOUS + _spSPLINE_2 : GO.1
```

will execute a Spline 2 move for axis 1 taking the positions in the spline table as true absolute motor positions. The spline will execute indefinitely. Once the last segment has been executed, the spline motion will begin again.

An *absolute spline* only defines absolute positions within one spline cycle and not to an absolute motor position. An absolute spline implies a zero at the start of each cycle. A *true absolute spline* takes the table values as absolute to true motor position.
The profile produced is:

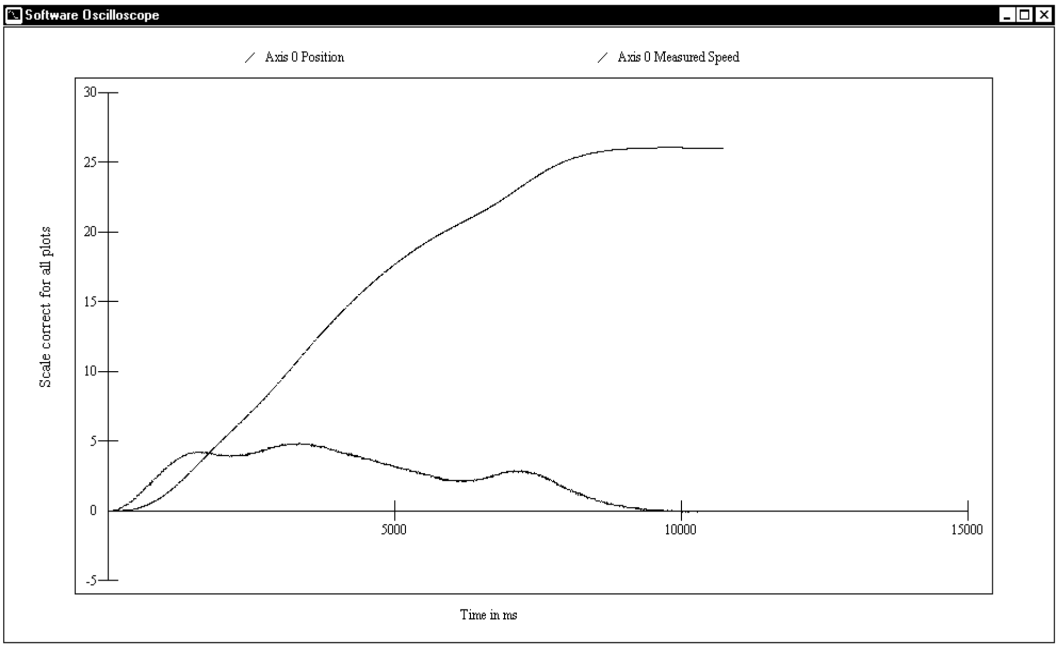


Figure 1: Example of spine move

Example:

An application requires that a liquid is moved smoothly from one point to another. If the STOP input is activated then come to a smooth stop.

```
REM Set up position and duration tables
DIM mypos(20) = 8,  1,   3,    7,   13,   17,   23,   25,   26
DIM mydur(20) =   500, 750, 1000, 1000, 2000, 1000, 1000, 1500
STOPSWITCHMODE = 1                : REM Pause if stop input activated
SPLINETABLE (0, mypos, NULL, mydur): REM The arrays to use
SPLINESUSPENDTIME = 2000          : REM The ramp time
SPLINE = _spABSOLUTE + _spSPLINE_1 : REM Spline 1 and absolute
GO                                : REM Start motion
```

Alternatively, the Spline 2 motion would produce:

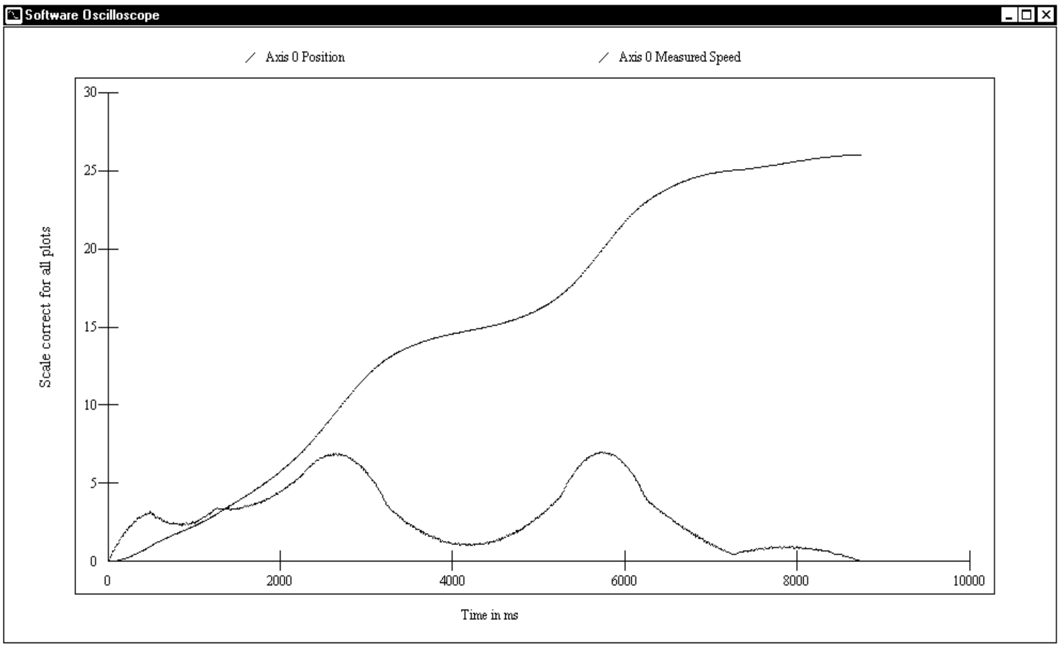


Figure 2: Example of spine move

It can be seen that the Spline 1 motion is smoother but the Spline 2 follows the position points. The Spline 1 move also takes slightly longer. This is because the first and last segments are doubled in duration.

When a spline move is stopped, the axis will produce a controlled stop over a duration defined with the **`SPLINESUSPENDTIME`** keyword.

The **`SPLINESTART`** and **`SPLINEEND`** keywords can me used to modify the segments used in a spline table.

3.3 Blending

This section applies only to the NextMove product range.

Normally, in order to change between two different move types, the axis must be stopped first and allowed to become idle. Blending is where the transition between move types can be done on the fly, without the axis being forced to come to a halt. It is possible to blend into a **`JOG`** move or a **`TORQUE`** move from any other move type with the exception of a **`HOME`** cycle or a linear or circular interpolated move.

When blending to a **`JOG`** move, the axis will ramp from its current speed to the new **`JOG`** speed using the current **`ACCEL`**, **`DECEL`** values.

3.4 Compensation Modes

This section applies only to the NextMove product range.

There are a number of compensation modes available in Mint; Backlash compensation, Leadscrew compensation.

3.4.1 Backlash Compensation

Backlash describes the amount of free movement in the gearbox or belts linking a motor and the physical axis. Backlash compensation automatically compensates for this free movement whenever the axis changes direction. The compensation method used is known as constant compensation and is turned on with the **`BACKLASHMODE`** keyword.

The size of the backlash is set with the **BACKLASH** keyword, the sign of the backlash is used to indicate in which direction the backlash was taken up during the homing cycle.

Example:

```
BACKLASH[0,1] = 0.01, -0.2
```

This indicates that backlash on axes 0 and 1 is 0.01 and 0.2 user units respectively.

The rate at which compensation is applied is controlled with the **BACKLASHINTERVAL** keyword. As the axis changes direction, it will take the specified number of servo cycles to apply the compensation size.

3.4.2 Leadscrew Compensation

Inaccuracies in the manufacture of leadscrews result in discrepancies between the theoretical and actual linear position of a nut. This error is called *Lead Error*. Leadscrew Compensation is a software function that attempts to correct Lead Error.

Many leadscrew manufacturers can provide a lead precision table for each leadscrew supplied. This table details the actual nut position for a number of theoretical positions along the usable length of the leadscrew. The positions are usually measured from one marked end of the screw and are taken at regular intervals. There may be a table for travel in both directions, effectively allowing for backlash as well.

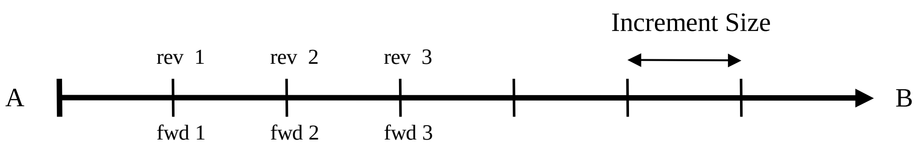
The controller stores similar tables for each axis. Each entry in a table will be the absolute actual position of the nut in user units. The table is referred to as a **PRECISIONTABLE**. It is possible to specify a uni-directional or bi-directional tables.

The theoretical position increment between consecutive entries is programmable and applies to the whole table. This is the **PRECISIONINCREMENT**.   The number of entries in a table will be programmable limited only by memory space. The compensation feature can be turned on and off.

If Leadscrew Compensation is turned on the measured positions will be modified by the values stored in the Precision Tables. A modified value will be linearly interpolated between two adjacent table entries. The table entries used are based upon the theoretical measured position and the **PRECISIONINCREMENT**. As the home position may not coincide with the start of the table, a position offset must be added to the measured position prior to table lookup. This is the PRECISIONOFFSET.

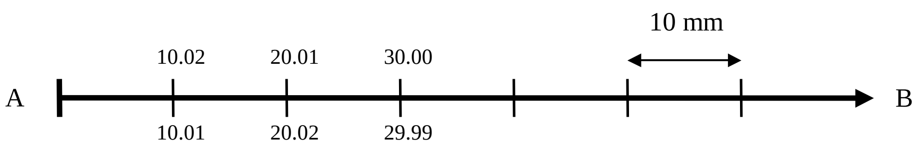
Leadscrew Compensation will be turned off automatically if the position falls outside the range of the *Precision Table*.   It will be automatically turned back on when the position falls within the range of the table.

Leadscrew compensation assumes that the leadscrew can be described as:



where *fwd x* are the actual values for travel in direction from A to B and *rev x* are the actual values for travel in the direction from B to A. For single table compensation, the *fwd x* values would apply in both direction.

For a dual table leadscrew, the values might be




```
DIM fwdTable (5) = 10.02, 20.01, 30.00, 40.01, 50.01
DIM revTable (5) = 10.01, 20.02, 29.99, 40.00, 50.01

PRECISIONTABLE(0, fwdTable, revTable) : REM Forward and reverse tables
```

```
PRECISIONINCREMENT.0 = 10            : REM 10mm table steps
PRECISIONOFFSET.0 = 0                : REM Assume start of table is home
PRECISIONMODE.0 = _lsDUAL_TABLE      : REM Turn on dual table compensation
```

3.5 Hold To Analog

This section applies only to the NextMove product range.

The purpose of the HTA is to keep the value on an analog input constant where the position of an axis has an effect on that analog value. Once the controller is put in HTA mode, position is controlled in order to keep the analog value constant.

HTA effectively adds a force control loop around the outside of the standard servo loop. Figure 3 shows a block diagram of the force control system. The output of the force controller is a position demand signal which is applied to NextMove's standard position controller. The transfer function of the position control loop is therefore $G_{c2}(s)=1$.

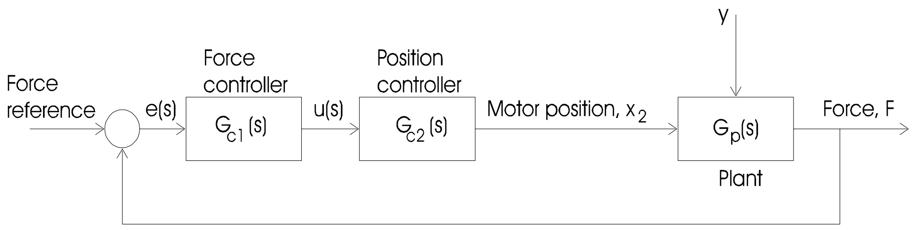


Figure 3: Flow of Force Control System

In order to satisfy a majority of plant models, the force control loop implemented is as follows:

$$Velocity = Error \times Gain + \Delta Error \times Damping$$

where:

Velocity	is the demand signal passed to the positional control loop.
Error	is the difference between the desired analog value and the measured value.
ΔError	is the change in error from the current and previous sample.
Gain	is the proportional gain term.
Damping	is the damping term.

Within the force control loop, the analog input channel is sampled and filtered if required. The error between the measured and the desired value is then calculated and checked against the error deadband. If the error is not within the deadband then a velocity demand will be generated.

The velocity demand is then limited within the currently defined motion trapezoid. This means that the axis acceleration or deceleration will be limited to the currently set values and the maximum velocity will be limited to the currently set speed value.

3.5.1 HTA Setup

In order to perform HTA, an analog input channel has to be associated with an axis. The force control loop must then be tuned and the analog filter and deadband used to control sensitivity. The motion profile is defined to control the speed and acceleration of the axis as required.

The force control algorithm is essentially error driven. It will try to minimize any error between the current analog input channel value and the user specified analog hold value

There are a number of keywords associated with HTA:

Mint Keyword	Use
HTA	Starts the HTA mode of motion and specifies the desired hold value.
HTACHANNEL	Associates an analog input channel with an axis.
HTAGAIN	The proportional gain in the force control loop.
HTADAMPING	The damping term in the force control loop.
HTAFILTER	The factor for the analog input channel filter.
HTADEADBAND	The analog error deadband.

Example:

```
REM Initialize HTA parameters.  Axis 0 will use analog channel 0 to try and
REM hold to a value passed from the host PC via the comms protocol.
AXES[0]
HTACHANNEL = 0     : REM Using analog channel 0
HTAKPROP = 0.1     : REM HTA gain term
HTADAMPING = -0.05 : REM HTA damping term
HTAFILTER = 0.999  : REM High filter value
HTADEADBAND = 5    : REM Deadband of 5 analog counts
SPEED = 10         : REM Maximum slew speed
ACCEL = 100        : REM Maximum acceleration rate
DECEL = 100        : REM Maximum deceleration rate

REM COMMS 1 controls when to go into HTA.  COMMS 2 is the desired value.
WHILE (1)
  IF COMMS(1) = 1 DO
    HTA = COMMS(2) : REM Hold at the value specified by COMMS(2)
  ELSE
    STOP
  ENDIF
ENDW
```

Advanced use of Fast Position Latching

4

This chapter details the advanced features associated with fast position latching, specifically:

- ◊ Mapping inputs to axes.
- ◊ Fast position latch modes.

In order to utilize the functionality of the ASIC's hardware position latching facility the functionality of fast position latching has been modified.

Each controller supports a different number of axes and fast interrupt inputs:

Controller	Fast Interrupt Inputs	Axes	Auxiliary Encoders
NextMove BX	4	4	1
NextMove PC	1	8	0
NextMove PCI[2] (main board only)	4	8	1
NextMove PCI (main board + 1 expansion card)	8	8	2
MintDrive	2	1	1
ServoNode 51	2	1	1

Although the hardware available differs between controllers the keywords and functionality is the same across all controllers. The only functional difference being that the controllers which use the encoder ASIC (NextMove PCI, MintDrive and ServoNode 51) perform their position capture within hardware and are consequently much faster.

On NextMove PCI, in order to latch position of a stepper axis, the ASIC must be put into the correct mode. Each ASIC is capable of supporting one servo axis and one stepper axis. Each axis is associated with a channel and this can be read with the **AXISCHANNEL** keyword. The **VIEW CONFIG** keyword can also be used to display this information in the terminal.

A channel (ASIC) can be switched to allow capture of the servo or stepper axis that is using that channel. This is done with the **CHANNELCONFIG** keyword. This must be done before attempting to call any of the FASTxxx keywords otherwise an error 'Incorrect Channel Setup' will be returned indicating that the channel an axis is using is not setup to allow capture on that axis.

The flexibility of the fast position latch capability within Mint v4 has been expanded, however this means that the setup of the axis and the inputs may need to be configured.

The **FASTSELECT** keyword allows axes to be assigned a fast position latch input. When this assigned input becomes active the position of all associated axes is latched. For Example:

```
FASTSELECT[0,1,2,3] = 1,1,2,3
```

will map: axis 0 to input 1

axis 1 to input 1

axis 2 to input 2

axis 3 to input 3

By default all axes (including the auxiliary encoders) are mapped to fast position latch input 0.

In the case of NextMove PCI all axes on the expansion card are mapped to input 20.

The **FASTLATCHMODE** keyword sets the mode of the position latching. This controls when the axis position can be latched. The available modes are:

- 0 – Always
- 1 – Smart
- 2 – Manual
- 3 – Never (Off)

In mode 0: The latch preventing further position capture is cleared as soon as the position has been captured.

In mode 1: Further latching is prevented until the fast position latch handler has completed.

In mode 2: Further latching is prevented until the user manually clears the hardware latch using the **FASTENABLE** keyword.

In mode 3: The position capture is disabled.

[2] Fast position latching is only supported on the servo axes of a NextMove PCI controller

In the event of a fast position latch, a handler will be called to enable the user to deal with this event. Within Mint the user can install a handler for each input (**#FASTINx**) where x is the digital input number that causes the latch.

On MintDrive and ServoNode 51, the following handlers are possible:

- **#FASTIN0** – input 0
- **#FASTIN2** – input 2 (only latches the auxiliary encoder)

On NextMove PC, the following handlers are possible:

- **#FASTIN0** – input 0

On NextMove PCI and NextMove BX, the following handlers are possible:

- **#FASTIN0** – input 0
- **#FASTIN1** – input 1
- **#FASTIN2** – input 2
- **#FASTIN3** – input 3

Adding an expansion card to NextMove PCI adds the following handlers:

- **#FASTIN20** – input 20
- **#FASTIN21** – input 21
- **#FASTIN22** – input 22
- **#FASTIN23** – input 23

The **FASTENABLE** keyword clears the hardware latch to enable further position latching to occur. This is only applicable if using **FASTLATCHMODE** 2.

The **FASTLATCH** keyword will return 0 or 1 indicating if an axis has latched position or not. The **FASTENCODER** keyword will return the latched encoder value for an axis. The **FASTPOS** keyword will return the latched position value for an axis.

Alternatively, a single event handler can be used called **#FASTIN** but this is not recommended where different latch modes are being used on different inputs. If using the **#FASTIN** handler then the **FASTLATCH** keyword must be read in order to clear the latch on each axis.

The above keywords (with the exception of **FASTPOS**) are duplicated for the auxiliary encoders. The functionality is the same, the only difference being they apply to the auxiliary encoder rather than the controlled axes. For example, **FASTAUXLATCHMODE**, **FASTAUXENABLE** etc.

Example

Multiple handlers

```
FASTSELECT.0 = 0 : REM axis 0 will use input 0 to latch position
FASTSELECT.1 = 0 : REM axis 1 will use input 0 to latch position
FASTSELECT.2 = 3 : REM axis 2 will use input 3 to latch position
FASTSELECT.3 = 2 : REM axis 3 will use input 2 to latch position

FASTLATCHMODE[0,1,2,3] = 1; : REM re-latch only when the handler is complete

FASTENABLE[0,1,2,3] = 0;    : REM clear all latches


PAUSE INKEY                 : REM wait for key press
END                         : REM end of program

REM
REM fast interrupt 0
REM
#FASTIN0
  ?”Input 0 Latched”
```

```
  ?FASTPOS.0                : REM latch position of axis 0
  ?FASTPOS.1                : REM latch position of axis 1
RETURN

REM
REM fast interrupt 2
REM
#FASTIN2
  ?"Input 2 Latched"
  ?FASTPOS.3                : REM latch position of axis 3
RETURN

REM
REM fast interrupt 3
REM
#FASTIN3
  ?"Input 3 Latched"
  ?FASTPOS.2                : REM latch position of axis 2
RETURN
```

Single handler

```
DIM i = 0

FASTSELECT.0 = 0 : REM axis 0 will use input 0 to latch position
FASTSELECT.1 = 0 : REM axis 1 will use input 0 to latch position
FASTSELECT.2 = 3 : REM axis 2 will use input 3 to latch position
FASTSELECT.3 = 2 : REM axis 3 will use input 2 to latch position

FASTLATCHMODE[0,1,2,3] = 1; : REM re-latch only when the handler is complete

FASTENABLE[0,1,2,3] = 0;    : REM clear all latches


PAUSE INKEY                 : REM wait for key press
END                         : REM end of program

REM
REM fast interrupt handler
REM
#FASTIN
  FOR i = 0 TO 4
    IF FASTLATCH.i THEN ?"Axis ", i, " latched : ", FASTPOS.i
  NEXT
RETURN
```

Advanced Error Handling

5

This chapter details advanced error handling within Mint.

- ◊ Changing the type of action taken in the event of an axis error
- ◊ Miscellaneous errors
- ◊ Axis warnings

5.1 Changing the Default Action of Motion Errors

There are 10 asynchronous axis motion errors. These are indicated by a bit pattern which can be read using the **AXISERROR** keyword. Each type of motion error is represented as follows:

Bit	Meaning
0	Motion aborted
1	Forward hardware limit reached
2	Reverse hardware limit reached
3	Forward software limit reached
4	Reverse software limit reached
5	Fatal following error limit exceeded
6	*Reserved*
7	External error input active
8	*Reserved*
9	*Reserved*
10	ADC limit exceeded
11	Master/Slave synchronization error
12	*Reserved*
13	*Reserved*
14	*Reserved*
15	*Reserved*
16	*Reserved*
17	Velocity limit exceeded

All of the above error conditions[3] have associated 'modes' which specify the action to be taken by the controller in the event of such a condition. These are set-up using the following keywords:

Mint Keyword	Meaning
ABORTMODE	Controls the action taken in the event of an abort.
LIMITMODE	Controls the action taken in the event of a hardware limit being reached.
SOFTLIMITMODE	Controls the action taken in the event of a software limit being reached.
FOLERRORMODE	Controls the action taken in the event of the maximum follow error being exceeded.
ERRORINPUTMODE	Controls the action taken in the event of the external error input becoming active.
ADCERRORMODE	Controls the action taken in the event of an analog input exceeding its specified limit.
VELFATALMODE	Controls the action taken in the event of a fatal velocity error occurring.

The possible modes are as follows:

Mode	Action
0	Ignore the error condition.
1	Crash stop the axis and drop the enable line. When the axis is idle[4] the error handler will be called.
2	Crash stop the axis, leave the axis enabled. When the axis is idle the error handler will be called.

[3] With the exception of a master/slave synchronisation error.

[4] An axis is deemed to be idle if it is disabled or if it has a following error.

Mode	Action
3	Perform a controlled stop on the axis at the rate specified by the **ERRORDECEL** parameter, leave the axis enabled. When the axis is idle the error handler will be called.
4	The forthcoming software limit is anticipated. The forward or reverse software limit becomes a temporary stop point and the motion will decelerate at the rate specified by **DECEL** to stop on the software limit. The axis will remain enabled. When the axis is idle the error handler will be called. (**Software Limit only**).
5	Call the error handler only.
6	Ramp the DAC to zero with a rate set with the **DACRAMP** parameter. The axis will be disabled when the DAC reaches zero and the error handler will be called.

Each of the errors for each of the axes can be given a separate mode and the controller will react accordingly.

As the above table details, the error handler will be called once the required action has been completed. However this may not always be required and the **ERRORMASK** parameter allows the user to specify whether or not a specific error on an axis will call the error handler. By default all errors will call the error handler on completion of the required action.

The keyword **ERRORMASK** accepts a bit pattern where if the bit is set the error handler will be called when that error condition becomes active. The bits are the same as for the axis error pattern listed above.

The table below shows the possible error modes for each of the different types of errors.

	Mode						
Error	**0**	**1**	**2**	**3**	**4**	**5**	**6**
Abort	✓	✓	✓	✓	×	✓	×
Hardware Limit	✓	✓	✓	✓	×	✓	×
Software Limit	✓	✓	✓	✓	✓	✓	×
Following Error	✓	✓	×	×	×	✓	✓
External Error	✓	✓	✓	✓	×	✓	×
Analog Error	✓	✓	✓	✓	×	✓	×
Slave/Sync Error[5]	×	✓	×	×	×	×	×
Velocity Error	✓	✓	✓	✓	×	✓	✓

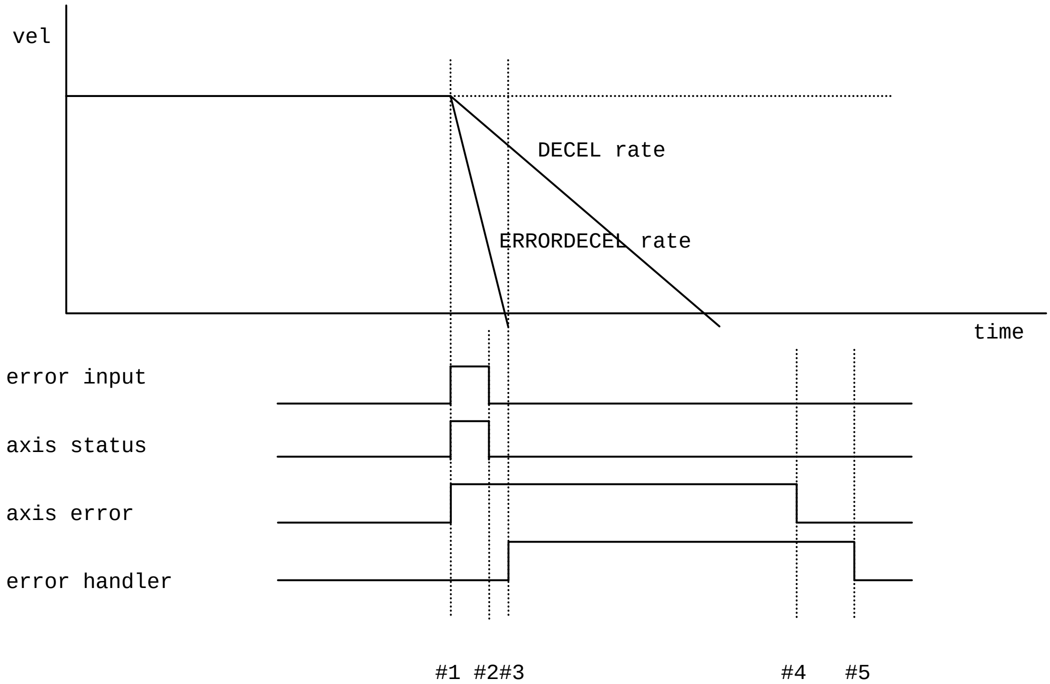
Every 2ms all of the potential error sources are checked. If any of them are true the axis status is immediately flagged and the appropriate action is initiated. Upon completion of the desired action, if the error mask allows and one exists, the error handler will be called.

The Axis Status flag is a real time indication of the status of the axis, for example if an error input is pressed, the axis status will reflect the state of the error input and when it is released the axis status will reflect this even if the error produced by this event is still being handled.

The Axis Error flag is a latched bit pattern of errors that have been handled, the axis error flag will on be set when the error occurs and will remain set until cleared by either canceling the errors using the **CANCEL** keyword, or manually clearing a specific bit by writing to the axis error bit pattern using the **AXISERROR** keyword

Below is an crude 'timing' diagram showing the change in state of the various flags in response to an error input. The mode for the axis is set up as mode 3 (decelerate using the **ERRORDECEL** rate).

[5] Not user selectable



#1 : Error input activated
#2 : Error input deactivated
#3 : Desired action completed and error handler called
#4 : Error cancelled
#5 : Error handler completes

While an error is present the handler will continue to be called every 2ms.

In previous versions of Mint, in the event of an asynchronous error Mint has always dropped the relay. This functionality has been removed and any changes to the outputs are controlled by the Mint Motion Library.

In order to maintain backwards compatibility in functionality a new function has been introduced which allows the user to specify a global error output with will be dropped in the event of an error. This output can be any of the digital outputs including the relay.

This error output is set using the keyword **GLOBALERROROUPUT**.

5.2 Miscellaneous Errors

Miscellaneous errors are non axis dependant asynchronous error conditions. They are represented as a bit pattern as follows:

Bit	Symbol	Meaning
0	me12V	12 volt rail failure
1	meOUTPUT_FAULT0	Output short circuit for bank 0
2	meOUTPUT_FAULT1	Output short circuit for bank 1
3	meOUTPUT_FAULT2	Output short circuit for bank 2
4	meRS232_RECEIVE_OVERRUN	RS232 receive overrun
5	meRS232_TRANSMIT_OVERRUN	RS232 transmit overrun
6	meRS232_RECEIVE_ERROR	RS232 receive framing error
7	meCAN_RECEIVE_OVERRUN	CAN serial redirection receive overrun
8	meSERVO_TICK_OVERRUN	The servo loop or profiler have exceeded their timeslice
9	meRS485_RECEIVE_OVERRUN	RS485 receive overrun
10	meRS485_TRANSMIT_OVERRUN	RS485 transmit overrun
11	meRS485_RECEIVE_ERROR	RS485 receive framing error

12	meOUTPUT_POWER0	Output driver not powered or missing for bank 0
13	meOUTPUT_POWER1	Output driver not powered or missing for bank 1
14	meOUTPUT_POWER2	Output driver not powered or missing for bank 2

Reading miscellaneous errors returns the bit pattern shown above. Miscellaneous errors are latched and will remain set until cleared by the user. They can be cleared by writing to **MISCERROR** with a value of zero.

Errors cannot be cleared individually.

They are also cleared on a call to the **CANCEL** or **RESET** functions.

If a 12V fault occurs (bit 0), all axes are disabled since the DACs and ADCs will not operate correctly if the 12V rail is not at the correct voltage.

If an output fault occurs (bits 1, 2 or 3), all outputs in the bank that has the fault will be inactive. Default action in this case will be to clear any drive enable flags on axes that use those outputs.

If a servo loop overrun is detected all motion is canceled and the **LOOPTIME** and **PROFILETIME** of the controller are reset to default values.

None of the other errors have any default action and these error conditions can be ignored. A miscellaneous error will cause the S2 LED to flash green on NextMove PCI. An asynchronous axis error will flash S2 red. If both occur at the same time, then the LED will alternate between the two.

Miscellaneous error will generate calls to the user installed error handler. This can be disabled using the keyword **MISCERRORDISABLE**.

In the error handler in MINT, **ERR** will be set to 501.

5.3 Axis Warnings

Axis warnings provide a method of asynchronous detection of abnormal states without any automatic corrective action. Detectable warnings currently include:

- Following error warning limit has been reached
- Servo loop integrator has reached warning limit
- Axis speed has reached maximum speed limit
- Invalid encoder transition

The keyword **AXISWARNING** is used to read the current axis warnings. This returns a bit pattern as follows:

Bit	Meaning
0	The axis following error has reached of exceeded the warning limit set by **FOLERRORWARNING**.
1	The axis integrator term reached the value set by **KINTLIMIT**.
2	The speed of an axis has reached the value set by **MAXSPEED**.
3	An invalid encoder edge transition has occurred

These warnings are latched until cleared by writing zero to **AXISWARNING** or by calling **CANCEL** or **RESET**.

In the event of an **AXISWARNING**, the user defined error handler will be called for warnings that are enabled with **AXISWARNINGDISABLE**.

In the error handler in MINT, **ERR** will be set to 502.

All axis warnings are disabled by default.

Keyword Reference Guide

6

This chapter presents, in alphabetical order, descriptions for all Mint keywords. Descriptions are broken down into:

- ◊ Purpose
- ◊ Format
- ◊ Syntax

6.1 Mint Keyword Syntax

The following syntactical rules are applied throughout the following keyword definitions:-

Case:

Words in capital letters are keywords (for example **FOLERRORFATAL**) and must be entered as shown, except that they can be entered in any combination of upper and lower case. For example:

```
folerrorfatal = 10
FolErrorFatal = 20
FOLERRORFATAL = 30
```

Angled Brackets:

You are asked to supply any items shown in lower case letters between angled brackets. Where the term <expression> or <condition> is used, this refers to a variable, constant or other valid numeric expression. For example:

```
a * b / c
a
POS < 100
```

Curly Brackets:

Items in curly brackets { } are optional. For the example above:

```
FOLERRORFATAL[axis] = <expression> {,<expression> ...}
```

Continuation Dots:

The dots "..." signify that more expressions or statements can follow.

6.2 Mint Keyword Definitions

Each keyword is defined using the structure described below. Where sections are not applicable to a particular keyword they may be omitted.

The † symbol denotes that this keyword has changed in its implementation in Mint v4. For details see the Mint v4 Code Analyzer Tool Guide.

Purpose

The ‘Purpose’ section briefly summarizes the function of the keyword.

Controllers Supported

The ‘Controllers Supported’ list indicates which Controllers support the keyword.

Format

The ‘Format’ section details the syntax that may be used with the keyword. It also indicates if the keywords is axis or channel based.

Dot Parameters

The ‘Dot Parameters’ specifies what the keyword addresses, such as an axis or a channel. Use the table below to find the range of this parameter on the different controllers.

Parameter	NextMove PCI	NextMove PC	NextMove BX	Servo Node 51	MintDrive
Axis number	$0 \le x \le 11$	$0 \le x \le 7$	$0 \le x \le 7$	$x = 0$	$x = 0$
CAN Bus number	$1 \le x \le 2$	$1 \le x \le 2$	$1 \le x \le 2$	$1 \le x \le 2$	$1 \le x \le 2$
Capture channel	$0 \le x \le 5$	$0 \le x \le 5$	$0 \le x \le 5$	$0 \le x \le 5$	$0 \le x \le 5$
Digital input channel	$0 \le x \le 19$	$0 \le x \le 23$	$0 \le x \le 15$	$0 \le x \le 5$	$0 \le x \le 17$
Digital output channel	$0 \le x \le 11$	$0 \le x \le 11$	$0 \le x \le 7$	$0 \le x \le 2$	$0 \le x \le 8$
ADC channel	$0 \le x \le 11$	$0 \le x \le 7$	$0 \le x \le 7$	$x = 0$	$0 \le x \le 3$

DAC channel	0 ≤ x ≤ 11	0 ≤ x ≤ 7	0 ≤ x ≤ 3	N/A	N/A
Auxiliary DAC channel	N/A	N/A	N/A	0 ≤ x ≤ 1	0 ≤ x ≤ 3
Encoder channel	0 ≤ x ≤ 11	0 ≤ x ≤ 7	0 ≤ x ≤ 3	x = 0	x = 0
Auxiliary encoder channel	0 ≤ x ≤ 2	x = 0	x = 0	x = 0	x = 0
Keypad channel	8,16,32,64	8,16,32,64	8,16,32,64	8,16,32,64	8,16,32,64

Attributes

The ‘Attributes’ table is used to indicate the specific attributes of the keyword as described below:

Controller Indicates the Controller to which the attributes in the adjacent columns apply.
Read If the keyword can be read
Write If the keyword can be written to
Command If the keyword is a command
Multi-Axis If the keyword is axis based and can be applied to more than one axis at once either explicitly or through the AXES keyword.
Scaled If the values written to or read from the keyword are scaled by a scale factor
Default The default value for a keyword
Range Numeric range of the keyword. Please note that this range is for the value multiplied by the scale factor if the Scaled box is checked. If the range specifies a decimal part (.0) then the keyword will accept fractional values.

Description

The ‘Description’ section details the functionality of the keyword.

Example

The ‘Examples’ section is used to provide one or more examples of the keyword use.

Restrictions

The ‘Restrictions’ section details any restrictions that are placed on the keyword - for instance modes under which it can be used.

Controller Specifics

The ‘Controller Specifics’ section lists any differences that are specific to a particular controller, that are not indicated in the attributes table.

See Also

The ‘See Also’ section lists keywords that are related, alternatives to or can be used for related set-up purposes.

6.3 Mint Keywords

ABORTMODE/ABM

Purpose:

Controls the default action taken in the event of an abort.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
ABORTMODE[axes] = <expression> {,<expression> …}
v = ABORTMODE[axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●		●		1	$0 \le x \le 6$

Description:

Sets the default action to be taken when a ‘software abort’ motion error is generated.

The following modes are available.

Mode	Action
0	Ignore. The axis will be unaffected and a motion error is not generated if an abort condition exists. The **AXISSTATUS** keyword will indicate the abort condition.
1	Performs a crash stop on the axis and deactivates the drive enable output and drops the enable relay.
2	Performs a crash stop on the axis. The drive and enable relay are left enabled.
3	Performs a controlled stop on the axis. The axis will decelerate at the **ERRORDECEL** rate. The drive and enable relay are left enabled.
4	Not applicable
5	The error handler is called.
6	Not applicable

Example:

```
AXES[0,1,2]
ABORTMODE = 2;  : REM set default action
LOOP
  IF INKEY = ‘E’ DO
    ABORT
    EXIT
  ENDIF
ENDL
```

The above code will sit in a **LOOP..ENDL** until the key ‘E’ is pressed, when it will generate an abort error on all axis in the axes string. In this can axes 0, 1 and 2.

As **ABORTMODE** is set to 2, when the abort error occurs the axes will crash stop any motion but the drive enable will not be dropped.

See also:

ABORT, #ONERROR

ADCERROR/AE

Purpose:

Read back the analog channels currently in error.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
v = ADCERROR[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●			●			$0 \le x \le 65535$

Description:

It is possible to attach limits to analog input channels such that if those limits are exceeded, the motion error **_erADC_ERROR** is generated. Analog channels can be associated with one or more axes and an axis can have multiple analog channels associated with it.

The **ADCERROR** keyword returns a bit mask indicating the currently active analog errors on the specified axis. It also indicates which analog limit, upper and/or lower, caused the analog error.

Upper and lower analog limits are set on an analog channel using the **ADCMAX** and **ADCMIN** keywords. Channels are associated with an axis using the **ADCMONITOR** keyword.

The returned bit pattern is shown below:

Bit	Meaning
0	Analog error on channel 0. Maximum limit exceeded.
1	Analog error on channel 1. Maximum limit exceeded.
2	Analog error on channel 2. Maximum limit exceeded.
3	Analog error on channel 3. Maximum limit exceeded.
4	Analog error on channel 4. Maximum limit exceeded.
5	Analog error on channel 5. Maximum limit exceeded.
6	Analog error on channel 6. Maximum limit exceeded.
7	Analog error on channel 7. Maximum limit exceeded.
8	Analog error on channel 0. Minimum limit exceeded.
9	Analog error on channel 1. Minimum limit exceeded.
10	Analog error on channel 2. Minimum limit exceeded.
11	Analog error on channel 3. Minimum limit exceeded.
12	Analog error on channel 4. Minimum limit exceeded.
13	Analog error on channel 5. Minimum limit exceeded.
14	Analog error on channel 6. Minimum limit exceeded.
15	Analog error on channel 7. Minimum limit exceeded.

Example:

```
PRINT ADCERROR.4
```

Display any analog channel errors for those channels attached to axis 4. If the result printed was 265, this would mean the maximum limit has been exceeded on channels 0 and 3 and the minimum limit has been exceeded on channel 0.

See Also:

AXISSTATUS, ERR, AXISERROR, ADCMAX, ADCMIN, ADCMONITOR

ADCERRORMODE/ADM

Purpose:

Controls the default action taken in the event of an ADC limit being exceeded on an associated channel.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
ADCERRORMODE[axes] = <expression> {,<expression> …}
v = ADCERRORMODE[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		1	$0 \le x \le 5$

Description

ADC limits are set up with the keywords **ADCMAX** and **ADCMIN** and channels are associated to axes using the **ADCMONITOR** keyword. The **ADCERRORMODE** keyword determines the action taken by the controller in the event of an ADC error being generated. Valid modes are as follows:

Mode	Action
0	Ignore the error condition.
1	Crash stop the axis and drop the enable line. When the axis is idle[6] the error handler will be called.
2	Crash stop the axis, leave the axis enabled. When the axis is idle the error handler will be called.
3	Perform a controlled stop on the axis at the rate specified by the **ERRORDECEL** parameter, leave the axis enabled. When the axis is idle the error handler will be called.
4	Not applicable
5	Call the error handler.
6	Not applicable

Example:

```
ADCERRORMODE.0 = 2
ADCMAX.2 = 50
ADCMIN.2 = -50
ADCMONITOR.0 = 4
```

This sets the upper limit on channel 2 to be 50% of the maximum value and the lower limit on channels 2 to be –50%. Axes 0 is set to monitor ADC channel 2. Therefore if the value on analog channel 2 reached greater than 50%, axis 0 would incur analog motion error and the axis would take the action specified by **ADCERRORMODE**. In this case, crash stop and leave the axis enabled.

See Also:

ADCERROR, ADCMAX, ADCMIN, ADCMONITOR

ADCMAX/AMX

Purpose:

Sets the upper analog limit value for the specified analog channel.

[6] An axis is deemed to be idle if it is disabled or if it has a following error.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
ADCMAX.channel = <expression>
v = ADCMAX.channel
```

Dot Parameters:

Channel – ADC Channel Number

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		100	-100 ≤ x ≤ 100

Description:

It is possible to attach limit values to analog input channels such that if those limits are exceeded, a motion error is generated. Analog channels can be associated with one or more axes and an axis can have multiple analog channels associated with it.

The **ADCMAX** keyword sets the upper limit value for an analog channel to a percentage of its maximum value. The value must also be greater than then lower limit value set with **ADCMIN**. The limit value cannot be set on an analog channel that is configured as off.

In order for an analog channel to generate a motion error, it must be attached to an axis. This is done with the **ADCMONITOR** keyword. If the analog value then exceeds the upper or lower limit, a motion error is generated on that axis and all other axes that are monitoring that analog channel.

In the event of a motion error, the axis is crash stopped and the drive and enable relay disabled. The user defined error handler, #ONERROR, will be called if it is defined.

The **ADCERROR** keyword can be used to see which analog channels have caused an analog motion error on an axis.

Example:

```
ADCMAX.0 = 50
ADCMAX.2 = 75
ADCMIN.0 = -50
ADCMIN.2 = -75
ADCMONITOR[0,3,5] = 5;
```

This sets the upper limit on channels 0 and 2 to be 50% and 75% respectively and the lower limit on channels 0 and 2 to be –50% and –75% respectively. Axes 0, 3 and 5 are set to monitor channels 0 and 2. Therefore if the value on analog channel 2 reached 51%, then axes 0, 3 and 5 would incur analog motion errors and reading the **ADCERROR** keyword for these axes would return a value 4, indicating analog channel 2, upper limit.

See Also:

ADCERROR, ADCMODE, AXISERROR, ADCMIN, ADCMONITOR

ADCMIN/AMN

Purpose:

Sets the lower analog limit value for the specified analog channel.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
ADCMIN.channel = <expression>
v = ADCMIN.channel
```

Dot Parameters:

Channel – ADC Channel Number

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		-100	$-100 \leq x \leq 100$

Description:

It is possible to attach limit values to analog input channels such that if those limits are exceeded, a motion error is generated. Analog channels can be associated with one or more axes and an axis can have multiple analog channels associated with it.

The **`ADCMIN`** keyword sets the lower limit value for an analog channel to a percentage of its maximum value. The value must also be less than then upper limit value set with **`ADCMAX`**. The limit value cannot be set on an analog channel that is configured as off.

In order for an analog channel to generate a motion error, it must be attached to an axis. This is done with the **`ADCMONITOR`** keyword. If the analog value then exceeds the upper or lower limit, a motion error is generated on that axis and all other axes that are monitoring that analog channel.

In the event of a motion error, the axis is crash stopped and the drive and enable relay disabled. The user defined error handler, #ONERROR, will be called if it is defined.

The **`ADCERROR`** keyword can be used to see which analog channels have caused an analog motion error on an axis.

Example:

See **`ADCMAX`**

See Also:

ADCERROR, ADCMODE, ADCMAX, ADCMONITOR, AXISERROR

ADCMONITOR/AMR

Purpose:

Specifies the analog channels that an axis will monitor for analog limit checking.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
ADCMONITOR[axes] = <expression> {,<expression> ...}
v = ADCMONITOR[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	$0 \leq x \leq 255$

Description:

It is possible to attach limit values to analog input channels such that if those limits are exceeded, a motion error is generated. Analog channels can be associated with one or more axes and an axis can have multiple analog channels associated with it.

If an channel being monitored exceeds either its upper or lower limit value, a motion error will be generated. The action taken in the event of an error being generated depends on the **`ADCERRORMODE`**, but by default the axis will be crash stopped and the drive disabled. The user defined error handler, #ONERROR, will be called if it is defined.

The upper and lower limit values are set for a channel using them **ADCMAX** and **ADCMIN** keywords.

The **ADCMONITOR** keyword accepts a bit pattern representing the analog channels to monitor as follows:

Bit	Meaning
0	Analog channel 0.
1	Analog channel 1.
2	Analog channel 2.
3	Analog channel 3.
4	Analog channel 4.
5	Analog channel 5.
6	Analog channel 6.
7	Analog channel 7.

An axis can monitor any number of channels and more than one axis can monitor the same channel.

An analog channel can only generate an error condition if it is being monitored. If a channel is not monitored, then it cannot generate an error.

When an analog error occurs on an axis, the **ADCERROR** keyword can be used to determine which of the monitored channels caused the error.

Example:

See **ADCMAX**

Example:

```
ADCMONITOR = ADCMONITOR & 011110111
```

To stop analog errors be generated, the axis must not be monitored. The example stops channel 3 being monitored and leaves all channels in their current state.

See Also:

ADCERROR, ADCMODE, ERR, ADCMAX, ADCMIN, AXISERROR

ASYNCERRORPRESENT/AEP

Purpose:

To determine whether an asynchronous error is present.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
v = ASYNCERRORPRESENT
```

Dot Parameters:

None.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●						$0 \leq x \leq 1$

Description:

If an asynchronous error is present then **ASYNCERRORPRESENT** will return **_true**, otherwise it will return **_false**.

An asynchronous error will be present if there is either a motion error on any axis, a miscellaneous error or an axis warning on any axis. Details of motion errors on specific axes can be read using the keyword **AXISERROR**. Details of any miscellaneous errors can be read using the keyword **MISCERROR**. Details of any axis warnings on specific axes can be read using the keyword **AXISWARNING**.

See Also:

AXISERROR, AXISWARNING, MISCERROR

AUXENCODERMODE/AEM

Purpose:

To make miscellaneous changes to the Auxiliary Encoders.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●			●	●

Format:

```
AUXENCODERMODE.channel = <expression>
v = AUXENCODERMODE.channel
```

Dot Parameters:

Channel – Auxiliary Encoder Channel

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove PCI	●	●		●		0	0 ≤ x ≤ 7
MintDrive & ServoNode 51	●	●				0	0 ≤ x ≤ 16

Description:

The **AUXENCODERMODE** keyword accepts a bitmap as follows:

Bit	Purpose	Description	NM	SN/ MD
0	Encoder direction	Can be used to reverse the count direction of the auxiliary encoder input channels, to overcome possible wiring problems. 0 – Normal 1 – Reversed	●	●
1	Encoder type	Can be used to select the encoder type: 0 – Differential line receiver 1 – Single-ended encoder	●	
2	Input source	Selects the input source: 0 – Encoder inputs A and B 1 – Step (A) and Direction (B)	●	
3	Virtual	Creates a virtual auxiliary encoder reference. The **AUXENCODERSPEED** keyword is used to set the reference.		●
4	Off	Turns off all auxiliary encoder processing to give an increase in servo loop execution speed.		●

Example:

Check the count direction (polarity) of the auxiliary encoder using the Mint WorkBench QuickWatch window. If the auxiliary encoder count decrements when the motor shaft is rotated clockwise, then the count direction may be reversed:

```
AUXENCODERMODE = AUXENCODERMODE OR 1
```

Note: Changing AUXENCODERMODE can affect position related keywords

See Also:

AUXENCODER

AUXENCODERSPEED

Purpose:

Specifies a virtual speed reference for the auxiliary encoder.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
			●	●

Format:

```
AUXENCODERSPEED.channel = <expr>
v = AUXENCODERSPEED.channel
```

Dot Parameters:

Channel – Auxiliary Encoder Channel.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
MintDrive	●	●				0	-8388607 ≤ x ≤ 8388607

Description:

Specifies a virtual speed reference for the Aux. Encoder.

Example:

The following configures the Aux. Encoder as a virtual axis and sets it running at 1000 counts/sec.

```
AUXENCODERMODE.0 = 8 : REM Configure the Aux. Encoder as virtual axis
AUXENCODERSPEED.0 = 1000
MASTERSOURCE.0 = _msAuxEncoder
FOLLOW = 1
```

See also:

AUXENCODERMODE

AUXENCODERZLATCH/AEZ

Purpose:

To read the state of the Auxiliary Encoders Z latch.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●			●	●

Format:

```
AUXENCODERZLATCH.channel = <expression>
v = AUXENCODERZLATCH.channel
```

Dot Parameters:

Channel – Auxiliary Encoder Channel.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●						0 or 1

Description:

The encoder ASIC which is fitted to the NextMove PCI, MintDrive and ServoNode controllers will latch the edge of an encoder Z pulse within hardware. Generally the width of an encoder Z pulse is too narrow to poll for reliably using software alone. When an encoders Z pulse is seen the occurrence of this is latched within the hardware. It is this latched value that the **AUXENCODERZLATCH** keyword reads. The action of reading this register clears it, hence when using the **AUXENCODERZLATCH** keyword to search for a Z pulse the value of **AUXENCODERZLATCH** must be read first to ensure that any existing hardware latch is cleared.

Example:

```
a = AUXENCODERZLATCH.0          : REM read and clear the aux encoder z latch
REPEAT                          : REM wait until the z pulse is seen
  REM do nothing
UNTIL AUXENCODERZLATCH.0 = 1
AUXENCODER.0 = 0                : REM zero the aux encoder position
```

The above code example will firstly clear the auxiliary encoder z latch, then wait until another z pulse occurs, at which point the position of the auxiliary encoder is set to zero.

See Also:

AUXENCODER

AXISCHANNEL/ACH

Purpose:

Allows user mapping of hardware to axis numbers.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
AXISCHANNEL[axes] = <expression> {,<expression> ...}
v = AXISCHANNEL[axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		-	$0 \le x \le 12$

Description

Each axis on NextMove is mapped to a hardware channel. See section 2 for a description of the channel numbers. The **AXISCHANNEL** keyword is used to read the channel that an axis is mapped to or to change the axis mapping.

The number of axes of control varies but there are always 12 addressable axes on NextMove PCI. The **VIEW PLATFORM** keyword can be used to view this information.

To change the hardware channel that an axis uses, the axis must first be configured off with the **CONFIG** keyword. **AXISCHANNEL** is then used to change the mapping. The act of changing the mapping does not mean that the channel is in use and it is possible to have multiple axes mapped to the same hardware. Only when the axis is configured as a servo or stepper does the hardware channel become used and no further axes may be configured to use the same hardware channel.

Example:

If a 4 axis NextMove is used with a 4 axis expansion card, then there will be 8 axes of control. By default, all axes would be configured as servos with the following mappings:

- Axis 0 uses Channel 0

- Axis 1 uses Channel 1
- Axis 2 uses Channel 2
- Axis 3 uses Channel 3
- Axis 4 uses Channel 4
- Axis 5 uses Channel 5
- Axis 6 uses Channel 6
- Axis 7 uses Channel 7

To configure axis 4 as a stepper axis, then the **CONFIG** keyword would be used:

```
CONFIG.4 = _cfSTEPPER
```

This would make axis 4 a stepper axis using the first pulse and direction outputs on the expansion card. If we want axis 4 to use the pulse and direction outputs of channel 0 on the main card:

```
CONFIG.4 = _cfOFF       : REM Axis must be configured off to change the mapping
AXISCHANNEL.4 = 0       : REM This is ok as multiple axes may map to a channel
CONFIG.4 = _cfSTEPPER   : REM Axis 4 is now a stepper using channel 0
```

However, it would not now be possible to make axis 4 a servo axis as the servo component of channel 0 is currently being used by axis 0.

```
CONFIG.4 = _cfSERVO
```

This would give the error 'Hardware channel required in use'

The **VIEW CONFIG** keyword is used to view the current axis configurations and mappings. If the above example code was used, the mappings would now look like:

- Axis 0 uses Channel 0
- Axis 1 uses Channel 1
- Axis 2 uses Channel 2
- Axis 3 uses Channel 3
- Axis 4 uses Channel 0
- Axis 5 uses Channel 5
- Axis 6 uses Channel 6
- Axis 7 uses Channel 7

To see what hardware types are available for a channel, the **CHANNELTYPE** keyword is used.

See Also

CHANNELTYPE, CONFIG

AXISWARNING/AW

Purpose:

Read any current axis warnings.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
AXISWARNING[axes] = <expression> {,<expression> ...}
v = AXISWARNING[axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	0 ≤ x ≤ 15

Description:

Axis warnings provide a method of asynchronous detection of abnormal states without any automatic corrective action. **AXISWARNING** accepts a bit pattern as follows:

Bit	Meaning
0	The axis following error has reached of exceeded the warning limit.
1	The axis integrator term reached the value set by **KINTLIMIT**.
2	The maximum speed for an axis (as set with **MAXSPEED**) has been exceeded.
3	An invalid encoder edge transition has occurred

In the event of an **AXISWARNING** the appropriate bit in will be set. This is latched until it is cleared by manually writing zero to **AXISWARNING**, or by calling **CANCEL** or **RESET**.

By default no action will be taken in the event of an axis warning occurring, by clearing the appropriate bit in **AXISWARNINGDISABLE** it is possible to force the occurrence of an axis warning to call the #ONERROR routine. In the error handler in the Mint keyword **ERR** will be set to 502 to indicate that an axis warning is present.

All axis warnings wil NOT call #ONERROR by default.

Example:

```
AXISWARNINGDISABLE = 13 : REM disable all but KINTLIMIT warning
.
.
.
#ONERROR
  IF ERR = 502 DO       : REM check for axis warnings
    IF (AXISWARNING.0 AND _awKINT_LIMIT) DO
      REM if KINTLIMIT warning occurs then reduce the acceleration
      REM of the axis
      ACCEL = ACCEL * 0.5
    ENDIF
    AXISWARNING.0 = 0 : REM clear the axis warning
  ENDIF
RETURN
```

The above example firstly allows only axis warnings indicating that the **KINTLIMIT** has been reached to call the error handler.

Within the error handler the value of **ERR** is checked, if an axis warning exists and is of the correct type the acceleration of axis 0 is reduced. Then the axis warning is cleared.

If the axis warning is not cleared it will continue to generate calls to the error handler.

See Also:

AXISWARNINGDISABLE, ERR

AXISWARNINGDISABLE/AWD

Purpose:

Allows individual axis warnings to be enabled and disabled.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
AXISWARNINGDISABLE[axes] = <expression> {,<expression> ...}
v = AXISWARNINGDISABLE [axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		15	$0 \leq x \leq 15$

Description:

This function accepts a bit pattern which controls which axis warnings will call the error handler.

Bit	Meaning
0	Set to disable following error warning
1	Set to disable KINT limit warning
2	Set to disable max speed warning
3	Set to disable invalid encoder edge transition warning

Setting the appropriate bit in **AXISWARNINGDISABLE** will stop the axis warning from generating a call to #ONERROR. The axis warning bit pattern will always be updated regardless of the value of **AXISWARNINGDISABLE**. This keyword allows the user to prevent certain warning conditions from calling the #ONERROR routine.

Example:

```
AXISWARNINGDISABLE = 01101 : REM disable all except KINTLIMIT warnings
```

The above example sets **AXISWARNINGDISABLE** so that only axis warnings indicating that the **KINTLIMIT** has been reached to call the error handler.

See **AXISWARNING**

See Also:

AXISWARNING

BACKLASH/BL

Purpose:

To set the size of the backlash present on an axis.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
BACKLASH[axes] = <expression> {,<expression> ...}
v = BACKLASH[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●	●	0	-8388607.0 - 8388607.0

Description:

The **BACKLASH** keyword controls the size of the backlash on an axis. Backlash describes the amount of free movement in the gearbox or belts linking a motor and the physical axis. Backlash compensation automatically compensates for this free movement whenever the axis changes direction. The compensation method used is known as constant compensation and is turned on with the **BACKLASHMODE** keyword. The rate at which compensation is applied is controlled with the **BACKLASHINTERVAL** keyword.

The sign of the backlash is used to indicate in which direction the backlash was taken up during the homing cycle. If the initial home direction was positive, then the backlash size should be specified as a positive value. If the initial home direction was negative then the backlash size should be specified as a negative value. It is recommended that homing is only performed on a switch and not the encoder index pulse when using backlash compensation.

Example:

```
BACKLASH[0,1] = 0.01, -0.2
```

This indicates that backlash on axes 0 and 1 is 0.01 and 0.2 user units respectively. Axis 0 has homed in a positive direction and axis 1 in a negative direction. If the first move after homing on axis 0 was in a negative direction, then no compensation would be necessary. If the move was in a positive direction, then the size of the backlash would need to be compensated for.

Reading axis position will show the compensated position. The axis must be **idle** in order to change the backlash value. The default backlash is zero.

See section 3.4 for further details on backlash compensation.

See Also:

BACKLASHINTERVAL, BACKLASHMODE

BACKLASHINTERVAL/BLI

Purpose:

To set the rate at which backlash compensation is applied.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
BACKLASHINTERVAL[axes] = <expression> {,<expression> ...}
v = BACKLASHINTERVAL[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		50	$1 \leq x \leq 8388607$

Description:

When an axis changes direction, the size of the backlash needs to be taken up. If this was done instantly then a clunk would be seen on the axis. The **BACKLASHINTERVAL** keyword allows the number of servo loop cycles to be specified over which the backlash compensation is applied.

In order to smoothly take up backlash, the compensation is applied over a number of servo loop cycles. The time it takes to compensate also depends on the servo loop frequency set with the **LOOPTIME** keyword. The default servo loop interval is 1 milli-second so a backlash interval of 1000 would imply a 1000 servo cycles and therefore a compensation time of 1 second. If the servo loop interval was 500 micro-seconds, then the compensation time would be 500 micro-seconds.

Example:

```
BACKLASH.0 = 0.02
BACKLASHINTERVAL.0 = 200
```

The backlash size of 0.02 will be taken up over 200 servo cycles. The axis must be IDLE in order to change the backlash interval. Backlash is only taken up whilst the axis is motion.

See Also:

BACKLASH, BACKLASHMODE, LOOPTIME

BACKLASHMODE/BLM

Purpose:

Controls the use of backlash compensation.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
BACKLASHMODE[axes] = <expression> {,<expression> ...}
v = BACKLASHMODE[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	0 ≤ x ≤ 1

Description:

Backlash compensation can be turned on and off with the **BACKLASHMODE** keyword. 0 is off, 1 turns compensation on. When compensation is turned on, the sign of the backlash compensation size is used to determine the direction in which backlash was taken up.

It is assumed that if a positive size is given, then the motor is at the negative extent of the backlash area, as if the axis had been homed in an initial positive direction. This implies that for a negative movement, no compensation would need to be applied and for a positive movement, the backlash size would have to compensate in order to move the physical axis. If the backlash size was negative then the opposite assumption applies.

When the axis changes direction, the compensation size is taken up over the number of servo cycles specified with the **BACKLASHINTERVAL** keyword. The compensation method is known as constant since once a change in direction has occurred, the compensation value is applied constantly.

Compensation is only applied whilst an axis is in motion. Therefore if a large number of servo cycles are specified but the change in direction is achieved with a small move distance, then it is possible that the whole compensation distance will not be taken up with the move. When the next movement is given, the remaining compensation will be taken up.

Reading the axis position will return the compensated position, i.e., the physical axis position and not the motor encoder position. Axis velocity does show the effect of the compensation distance being taken up.

Backlash compensation only has affect whilst the axis is configured as a servo axis. It also has no affect if the axis is performing a **TORQUE** move.

Example:

```
BACKLASH.3 = -0.3
BACKLASHINTERVAL.3 = 50
MOVER.3 = 0.5
GO.3
PAUSE IDLE.3
BACKLASHMODE.3 = 1
```

This sets up backlash compensation on axis 3 of 0.3 mm to be applied over 50 servo cycles. A small negative move is performed to ensure that the backlash has been taken up in a positive direction; equivalent to a negative direction home sequence.

The axis must be **`idle`** in order to change the backlash mode and the default backlash mode is off.

See Also:

BACKLASH, BACKLASHINTERVAL, CONFIG

BUSVOLTAGE/BV

Purpose:

To read the DC bus voltage on a MintDrive.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
			●	

Format:

```
value = BUSVOLTAGE
```

Dot Parameters:

None.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
MintDrive	●					-	0 – 460V

Description:

Reads the DC bus voltage on a MintDrive. This is useful on an external 24V MintDrive to check that the mains voltage is at the correct level before attempting motion.

If the bus voltage is below the bus voltage threshold, an error will be generated. This will be reported by the **`DRIVEFAULT`** keyword. The threshold below which an error is generated is controlled by the **`BUSVOLTAGETHRESHOLD`** keyword.

Example

On a MintDrive with a 240V AC input, the DC bus voltage can be read as follows:

```
PRINT BUSVOLTAGE
338
```

The DC bus voltage is calculated as follows:

DC voltage = AC voltage x $\sqrt{2}$

See also:

`BUSVOLTAGETHRESHOLD, DRIVEFAULT`

BUSVOLTAGETHRESHOLD/BVT

Purpose:

To read and write the DC bus voltage threshold on a MintDrive.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
			●	

Format:

```
BUSVOLTAGETHRESHOLD = <expression>
value = BUSVOLTAGETHRESHOLD
```

Dot Parameters:

None.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
MintDrive	●	●				-	0 – 460V

Description:

On MintDrive, if the DC bus voltage drops below the bus voltage threshold, an error will be generated. This will be reported by the **DRIVEFAULT** keyword. The threshold below which an error is generated is controlled by the **BUSVOLTAGETHRESHOLD** keyword. Setting a voltage threshold of zero turns off the error detection.

The DC bus voltage is the AC bus voltage x $\sqrt{2}$. The current bus voltage can be read with the **BUSVOLTAGE** keyword.

See also:

BUSVOLTAGE, DRIVEFAULT

CAMAMPLITUDE/CMA

Purpose:

To modify the amplitude of a cam profile

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
CAMAMPLITUDE[axes] = <expression> {,<expression>}
v = CAMAMPLITUDE[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●			0.0 - 8388607.0

Description:

The **CAMAMPLITUDE** keyword allows a cam profile to easily scaled to change to position range of the slave motion. The slave positions are simply multiplied by the amplitude value, based on the position within the current cam cycle.

Example:

```
DIM camPositions(11) = 10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
CAMTABLE (0, camPositions, NULL)
CAM = 5
GO
```

This performs a continuous cam absolute within a cycle. If the cam amplitude was changed to 2, **CAMAMPLITUDE** = 2, then this would be the same as changing the position table values to: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20.

The amplitude value does not have an immediate affect on the profile, it will only come into effect at the start of the next CAM cycle.

See also:

CAM, CAMEND, CAMINDEX, CAMSTART, CAMTABLE

CAMPHASE()

Purpose:

Allows a CAM profile to be shifted forwards or backwards over fixed number of CAM segments.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
CAMPHASE ( axis, start segment, number of segments, delta master distance )
```

Dot Parameters:

Axis - Axis No.
Start Segment – Segment number at which the CAM phase should start
No of Segments – Number of segments to phase over
Delta Master Distance – Amount to modify the master distance by

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove			●				1 ≤ x ≤ 65535

Description:

The **CAMPHASE** keyword allows a CAM profile to be phase shifted forwards or backwards. This phasing occurs over a fixed number of specified CAM segments, specified by *number of segments.* If the CAM is being executed in a positive direction the phasing will begin when the segment number specified by *start segment* is entered. Conversely, if the CAM is being executed in a negative direction the phasing will begin when the CAM enters the segment number specified by the sum of *start segment* and *number of segments*.

The phasing is achieved by modifying the master distance by a fixed amount specified by *delta master distance*. This has the effect of stretching or squashing the CAM profile depending on the sign of the master distance modified.

It is possible to set-up a **CAMPHASE** over a greater number of segments than are in the CAM profile.

If a **CAMPHASE** is in progress and the CAM start or end points are modified using the keywords **CAMSTART** or **CAMEND** this change is buffered until the phasing is completed.

Only one **CAMPHASE** can be pending or in operation at any one time.

Example:

```
DIM camPositions(21) = 20,10,20,30,40,50,60,70,80,90,100,
                       100,90,80,70,60,50,40,30,20,10
CAMTABLE (0, camPositions, NULL)
```

```
CAM = 0
MASTERDISTANCE = 10
GO
CAMPHASE(0, 2, 5, 1)
```

This will modify the CAM profile over 5 segments starting when the CAM enters segment 2 (if moving in a positive direction, the phasing will start at segment 7 if moving in negative direction). The phasing will modify the **MASTERDISTANCE** by 1 during the phasing operation.

See also:

CAM, CAMEND, CAMPHASESTATUS, CAMSTART, CAMTABLE

CAMPHASESTATUS/CPS

Purpose:

To get the state of the **CAMPHASE** for a specific axis.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
v = CAMPHASESTATUS[axis]
```

Dot Parameters:

Axis = Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●			$0 \leq x \leq 2$

Description:

Gets the state of a **CAMPHASE** operation on a specific axis. Only one **CAMPHASE** operation can be loaded at a time, in order to establish whether an axis is performing or has a **CAMPHASE** pending the keyword **CAMPHASESTATUS** can be used.
When a **CAMPHASE** is in progress the **CAMPHASESTATUS** will return a non zero value as defined in the table below.

Value	Meaning
0	No **CAMPHASE** in progress or pending, OK to load a **CAMPHASE** operation.
1	**CAMPHASE** operation pending.
2	**CAMPHASE** operation in progress.

Example:

```
PAUSE CAMPHASESTATUS.0 = 0
CAMPHASE(0, 2, 5, 1)
```

The above code will pause for the **CAMPHASESTATUS** to be zero (i.e. there is no **CAMPHASE** in pending or in progress). At this point it is OK to load a new **CAMPHASE** operation.

See Also:

CAMPHASE

CAPTURE/CP

Purpose:

Controls the operation of capture.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
CAPTURE = <expression>
v = CAPTURE
```

Dot Parameters:

None.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove & MintDrive	●	●					0 ≤ x ≤ 2
ServoNode 51	●	●					0 ≤ x ≤ 3

Description:

The **CAPTURE** keywords are used to configure and perform data capture from controllers. A number of internal parameters can be logged automatically for later upload and display by the Mint WorkBench.

There are six capture channels on the controllers. Capture channels are of different sizes on different controllers depending on the memory available:

Controller	NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
Number of data points	1000	250	1000	1000	1000

Data capture is performed at a servo loop level, this means that by default, data is captured every servo loop. The keyword **CAPTUREINTERVAL** specifies the frequency of the data capture.

Each of the capture channels may be configured to capture a specific parameter from a specific axis. The keyword **CAPTUREMODE** specifies the data which will be captured. The keyword **CAPTUREAXIS** specifies the axis the data will be captured from.

The **CAPTURE** keyword is used to start and stop the capture process. Capture takes a parameter that controls the operation of capture as follows.

Value	Pre defined Constants	Description
0	**_capSTOP**	Stop a capture in progress.
1	**_capSINGLE**	Start a single capture. When the buffer becomes full, the capture will automatically stop.
2	**_capWRAP**	Start a continuous capture. When the buffer becomes full, the oldest values are overwritten until the capture is stopped.
3	**_capHIGH_SPEED**	Start a high speed capture (ServoNode 51 only)

Reading the **CAPTURE** keyword returns the current capture state.

Bit	Description
0	1 if running in single shot capture.
1	1 if running in continuous capture.
2	1 if the buffer has wrapped.

If a single capture has completed, then **CAPTURE** will return 0, indicating that it had completed.

Once data capture has been started there are various methods of stopping it. The first and most straight forward is to use the **CAPTURE** keyword specifying a mode of zero. Alternatively the data capture can be stopped internally when a specific event happens. The keyword **CAPTUREEVENT** specifies what type of event will stop the data capture.

Example:

```
LOOPTIME = 2000
CAPTUREINTERVAL = 1
CAPTUREMODE.0 = _cpMEASURED_SPEED
```

```
CAPTUREAXIS.0 = 0
CAPTURE = _capSINGLE
```

The above MintDrive example will capture 2 seconds worth of data, read every servo tick (i.e. 2 ms per point for 1000 data points), of the measured speed taken from axis 0.

```
CAPTUREINTERVAL = 2
CAPTUREMODE.0 = _cpMEASURED_SPEED
CAPTUREAXIS.0 = 2
CAPTURE = _capWRAP
PAUSE IN1 = 1
CAPTURE = _capSTOP
```

The above example will start to capture measured speed from axis 2 every second servo loop. The capture has been set to wrap so that when the capture buffer fills up, the oldest data is overwritten with new data. The capture is then halted after input 1 becomes active. The buffer will hold up to 4 seconds of data, ending at the point when the input became active.

ServoNode51 has an additional mode known as high speed capture. This data capture is performed in the drive stage of the controller. There are 256 data points which allows approximately 8ms of data capture. Attempting to capture non-high speed data will result in an error message.

See also:

CAPTUREAXIS, CAPTUREEVENT, CAPTUREINTERVAL, CAPTUREMODE

CAPTUREAXIS/CPA

Purpose:

To read or set the axis on a capture channel.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
CAPTUREAXIS.channel = <expression>
v = CAPTUREAXIS.channel
```

Dot Parameters:

Channel - Capture Channel.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●				0	$0 \leq x < 6$

Description:

The keyword **CAPTUREAXIS** can be used to either set which axis is assigned to a capture channel or to read which axis is currently assigned to a capture channel.

Example:

```
CAPTUREAXIS.0 = 0
CAPTUREMODE.0 = 1
CAPTURE = 1
```

The above example will capture the measured speed from axis 0.

Restrictions:

CAPTUREAXIS is not applicable to **CAPTUREMODE** 8.

CAPTUREAXIS specifics the auxiliary encoder channel for **CAPTUREMODE** 9 and 10.

See also:

CAPTUREINTERVAL, CAPTUREMODE, CAPTURE

CAPTUREEVENT/CPE

Purpose:

Configures cyclic capture to trigger (stop) on an event.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Dot Parameters:

None.

Format:

```
CAPTUREEVENT = <expr>
v = CAPTUREEVENT
```

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●				0	0 ≤ x ≤ 1

Description:

A cyclic capture is set up using the **CAPTURE** keyword. The capture event is then used to stop the cyclic capture.

The following events are currently supported.

Value	Description
0	Disable capture events. Capture must be manually stopped.
1	Trigger on an asynchronous axis error.
2	Trigger on a stop input event.

Using **CAPTUREEVENTDELAY**, the trigger can be programmed to continue capturing data for a specified period after the event. Cyclic capture is set using the **CAPTURE** keyword.

Example:

The following will trigger (stop the cyclic capture) on detecting any asynchronous error whilst performing the programmed moves, with a post-trigger delay of 100 milliseconds:

```
CAPTUREEVENT = 1
CAPTUREEVENTDELAY = 100
CAPTURE = 2
LOOP
  GOSUB performmove
ENDL
```

See also:

CAPTURE, CAPTUREEVENTDELAY, CAPTUREEVENTAXIS

CAPTUREEVENTAXIS/CPX

Purpose:

Sets the axis to monitor for the capture trigger event.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
CAPTUREEVENTAXIS = <expr>
v = CAPTUREEVENTAXIS
```

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●				0	0 to 12

Description:

Cyclic capture can be configured to automatically trigger on a specified event using the **CAPTUREEVENT** keyword. The CAPTUREVENTAXIS keyword is used to specify which axis to monitor for the trigger event. The axis must be a valid axis number for the controller.

Example:

The following will trigger (stop the cyclic capture) on detecting any asynchronous error on axis 3 whilst performing the programmed moves, with a post-trigger delay of 100 milliseconds:

```
CAPTUREEVENT = 1
CAPTUREEVENTAXIS = 3
CAPTUREEVENTDELAY = 100
CAPTURE = 2
LOOP
  GOSUB performmove
ENDL
```

See also:

CAPTURE, CAPTUREEVENT, CAPTUREINTERVAL, CAPTUREMODE, CAPTUREAXIS

CAPTUREEVENTDELAY/CPD

Purpose:

Defines the post-trigger delay for event capture.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Dot Parameters:

None.

Format:

```
CAPTUREEVENTDELAY = <expr>
v = CAPTUREEVENTDELAY
```

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●				0	$0 \leq x \leq 1000$

Description:

Cyclic capture can be configured to automatically trigger on a specified event using the **CAPTUREEVENT** keyword.

Using **CAPTUREEVENTDELAY**, the trigger can be programmed to continue capturing data for a specified period after the event. Cyclic capture is set using the **CAPTURE** keyword.

Example:

The following will trigger (stop the cyclic capture) on detecting any asynchronous error whilst performing the programmed moves, with a post-trigger delay of 100 milliseconds:

```
CAPTUREEVENT = 1
CAPTUREEVENTDELAY = 100
CAPTURE = 2
LOOP
  GOSUB performmove
ENDL
```

See also:

CAPTURE, CAPTUREEVENT, CAPTUREINTERVAL, CAPTUREMODE, CAPTUREAXIS

CAPTUREINTERVAL/CPI

Purpose:

To define the interval of the data capture in relation to the servo frequency.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
CAPTUREINTERVAL = <expression>
v = CAPTUREINTERVAL
```

Dot Parameters:

None.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●				1	$0 < x \le 60$

Description:

The keyword **CAPTUREINTERVAL** is used to specify the interval of data capture reads in units of servo loop ticks. This does not apply to high speed capture on ServoNode 51.

Example:

```
LOOPTIME = 1000
CAPTUREINTERVAL = 4
CAPTUREAXIS.0 = 0
CAPTUREMODE.0 = 1
CAPTURE = 1
```

The above example will capture the measured speed from axis 0, every 4th iteration through the servo loop (i.e. every 4 ms).

See also:

CAPTUREAXIS, CAPTUREMODE, CAPTURE, LOOPTIME

CAPTUREMODE/CPM

Purpose:

To read or set the mode on a capture channel.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
CAPTUREMODE.channel = <expression>
v = CAPTUREMODE.channel
```

Dot Parameters:

Channel - Capture Channel.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●				0	see below

Description:

The keyword **CAPTUREMODE** can be used to either assign a capture mode for a capture channel or to read the mode currently assigned to the capture channel.

The valid modes of capture are as follows:

Value	Pre defined Constants	Data to capture
0	`_cpOFF`	Off
1	`_cpMEASURED_SPEED`	Measured Speed
2	`_cpFOLLOWING_ERROR`	Following Error
3	`_cpMEASURED_POSITION`	Measured Position
4	`_cpDAC_DEMAND`	DAC Demand
5	`_cpDEMAND_POSITION`	Demand Position
6	`_cpDEMAND_SPEED`	Demand Speed
7	`_cpMODE`	Mode
8	`_cpINPUTS`	Digital Inputs
9	`_cpAUX_ENCODER`	Auxiliary Encoder Position
10	`_cpAUX_ENCODER_SPEED`	Auxiliary Encoder Velocity
11	`_cpHS_MEASURED_CURRENT`	High Speed Measured Current (SN51 only)
12	`_cpHS_MEASURED_VELOCITY`	High Speed Measured Velocity (SN51 only)
13	`_cpHS_DEMANDED_CURRENT`	High Speed Demand Current (SN51 only)
14	`_cpHS_EFFORT`	High Speed Effort (SN51 only)
15	`_cpENCODER`	Encoder Value
16	`_cpENCODER_SPEED`	Encoder Velocity

Example:

```
CAPTUREAXIS.0 = 0
CAPTUREAXIS.1 = 0
CAPTUREMODE.0 = 1
CAPTUREMODE.1 = 3
CAPTURE = 1
```

The above example will capture the measured speed, from axis 0, on channel 0 and the measured position, from axis 0, on channel 1.

See also:

CAPTUREAXIS, CAPTUREINTERVAL, CAPTURE

CHANNELCONFIG/CHC

Purpose:

To select if an ASIC channel on NextMove PCI will capture servo or stepper position information.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●				

Format:

```
CHANNELCONFIG.channel = <expression>
value = CHANNELCONFIG.channel
```

Dot Parameters:

channel – the channel number.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●				1	$1 \leq x \leq 2$

Description:

On NextMove PCI, servo and stepper motion is performed by a ASIC which is capable of supporting both a servo axis and a stepper axis. Each ASIC is referenced by a channel number and NextMove PCI can have up to four channels labelled 0 to 3. Channels on a NextMove PCI expansion card are labelled 4 to 7. See section 2 for a full description of controller channels.

An ASIC is capable of supporting position capture and position compare but only for one of the axes that is using that ASIC. The **CHANNELCONFIG** keyword allows the user to select if the position capture and position compare functionality can be used by the servo or stepper axis using that channel (ASIC). To select the servo axis, **CHANNELCONFIG** is set to 1. To select the stepper axis, **CHANNELCONFIG** is set to 2. The same Mint constants that are used with the **CONFIG** keyword can be used.

Value	Predefined Constant	Meaning
1	_cfSERVO	The servo axis using this channel has access to position capture and compare.
2	_cfSTEPPER	The stepper axis using this channel has access to position capture and compare.

Each axis is associated with a channel and this can be read with the **AXISCHANNEL** keyword. The **VIEW CONFIG** keyword can also be used to display this information in the terminal.

By default, all channels are configured to allow the servo axis using that channel access to position capture and position compare functionality. **CHANNELCONFIG** should be called before attempting to call keywords related to position capture and compare. If the channel is not set up correctly, the error 'Channel incorrectly configured' will be returned.

Example 1

On a 2 axis NextMove PCI card we require axis 1 to be a stepper axis and to perform position capture. By default, axes 0 and 1 will be configured as servo axes. To change axis 1 to a stepper axis:

```
CONFIG.1 = _cfSTEPPER
```

To see which channel axis 1 is using the **AXISCHANNEL** keyword is used:

```
channel = AXISCHANNEL.1
```

By default, this will be 1. We now need to configure channel 1 to allow stepper access to position capture:

```
CHANNELCONFIG.channel = _cfSTEPPER
```

Example 2

On an 8 axis NextMove PCI card we require axes 4 and 5 to be stepper axes and to perform position capture. By default, axes 4 and 5 will be configured as stepper axes. To see which channel axes 4 and 5 are using, the **AXISCHANNEL** keyword is used and **CHANNELCONFIG** called to select stepper control on the channel.

```
channel = AXISCHANNEL.4
CHANNELCONFIG.channel = _cfSTEPPER
channel = AXISCHANNEL.5
CHANNELCONFIG.channel = _cfSTEPPER
```

By default in this system, axis 4 uses channel 0 and axis 5 uses channel 1. Note, by default, axis 0 also uses channel 0. It would not be possible to perform fast position capture on this axis without using the **CHANNELCONFIG** keyword to switch the channel from stepper to servo control.

Example 3

A 2 axis NextMove PCI card is used with a 4 axis expansion card. We require axes 0 and 1 to be servo axes and axes 4 and 6 to be stepper axes and to perform position capture on these axes. By default, axes 0 and 1 will be configured as servo axes and their associated channels will be configured for servo control so no action needs to be taken. Axes 4 and 6 will need to be configured as stepper axes and their associated channels configured for stepper control. The **AXISCHANNEL** and **CHANNELCONFIG** keywords are called to select stepper control on the channel.

```
CONFIG.4 = _cfSTEPPER
channel = AXISCHANNEL.4
CHANNELCONFIG.channel = _cfSTEPPER
CONFIG.6 = _cfSTEPPER
channel = AXISCHANNEL.4
```

```
CHANNELCONFIG.channel = _cfSTEPPER
```

By default in this system, axis 4 uses channel 4 and axis 5 uses channel 5.

See also:

AXISCHANNEL, CONFIG, FASTSELECT, COMPAREMODE

CHANNELTYPE/CHT

Purpose:

Read what hardware is available to a specific channel.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
CHANNELTYPE.channel = <expression>
v = CHANNELTYPE.channel
```

Dot Parameters:

Channel – Channel No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●					$0 \le x \le 2$

Description:

Each hardware channel on NextMove consists of a number of components. The **CHANNELTYPE** keyword is used to read the hardware available for a channel number.

Meaning	Value	Macro
DAC output is present	1	**_ctDAC**
Pulse / Direction outputs are present	2	**_ctPULSE_DIRECTION**
Encoder is present	4	**_ctENCODER**
PWM hardware is present (NextMove PC only)	8	**_ctPWM**

See section 2 for a description of the channel numbers. **CHANNELTYPE** can be used to see if the hardware required is present on a channel.

The following table shows the hardware that must be present for each possible axis configuration.

Configuration	Hardware Required
SERVO	DAC & Encoder
STEPPER	Pulse / Direction output
PWM[7]	PWM Pulse / Direction output

Example:

To configure an axis as a servo axis, a DAC and ENCODER must be present on the channel the axis is mapped to.

```
channel = AXISCHANNEL.10   : REM Read the channel axis 10 is mapped to
IF (CHANNELTYPE.channel AND (_ctDAC + _ctENCODER) = (_ctDAC + _ctENCODER)) DO
  CONFIG.10 = _cfSERVO     : REM Assumes that h/w is not in use
ELSE
  PRINT "Required hardware is not present"
ENDIF
```

[7] NextMove PC only.

See Also:

AXISCHANNEL, CONFIG

COMPARELATCH/CML

Purpose:

Reads the state of the position compare latch.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●				

Format:

```
COMPARELATCH[axes] = <expression> {,<expression>}
v = COMPARELATCH[axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●			$0 \le x \le 2$

Description:

Reads the state of the position compare latch and returns a bit pattern:

Bit	Meaning
0	State of position compare pin: 1 – active 0 – inactive.
1	The position of the axis has matched the position in compare register 0.
2	The position of the axis has matched the position in compare register 1.

Bits 1 and 2 are latched values and can be cleared by writing to compare latch. Bit 0 is the state of the compare pin and will not be affected by writing the value written to **COMPARELATCH**.

Example:

```
LOOP
  CLS
  a = COMPARELATCH       : REM store copy of the latch
  COMPARELATCH = 0       : REM reset all latched bits
  IF (a & 0x01) DO
    PRINT “Compare Output is ON”
  ELSE
    PRINF “Compare Output if OFF”
  ENDIF
  IF (a & 0x02) THEN PRINT “Axis position matched value in register 0”
  IF (a & 0x04) THEN PRINT “Axis position matched value in register 1”
  WAIT = 250
ENDL
```

The above code will monitor the state of the compare output and print up a text message if the compare output changes state.

See Also:

COMPAREMODE, COMPAREPOS

COMPAREMODE/CMD

Purpose:

Enables and disables the position compare on an axis.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●				

Format:

```
COMPAREMODE[axes] = <expression> {,<expression>}
v = COMPAREMODE[axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	$0 \leq x \leq 1$

Description:

Setting the compare mode of an axis to 1 will enable the position compare on an axis. The position compare will remain enabled until it is turn off by setting compare mode to zero.

Each encoder channel has a position compare output, this mapping is fixed within the hardware and is:

Encoder Channel	Digital Output
0	0
1	1
2	2
3	3

This is similar for any expansion boards, the bottom 4 digital outputs are mapped the encoder channels on that card. When these outputs are not under position compare control they can be used as normal outputs. When the position compare is enabled the associated output is no longer under user control and cannot be written to using the **OUT** command. Reading the **OUT** keyword will not report the state of any pin under position compare control.

Position compare functionality is only avaliable on servo axes

Example:

```
COMPAREPOS.0.0 = 10.5
COMPAREPOS.0.1 = 15.2
COMPAREMODE.0 = 1      : REM enable the position compare
```

The above code will setup both the upper and lower position compare registers for axis zero, then enable the position compare hardware. This puts the associated output under position compare control, from this point on the output will turn on between positions 10.5 and 15.2.

See Also:

COMPARELATCH, COMPAREPOS

COMPAREPOS/CMP

Purpose:

This function writes to the ASIC position compare registers.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●				

Format:

```
COMPAREPOS.axis.register = <expression>
v = COMPAREPOS.axis.register
```

Dot Parameters:

Axis – Axis No.
Register – Position compare register.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	-8388607.0 - 8388607.0

Description:

There are two position compare registers on the ASIC, register 0 and 1. The when the position of the axis matches the value held is a position compare register the compare pin will change state. The compare output will be active when the axis position is greater than the value in compare register 0 and less than the value in compare register 1.

See Also:

COMPARELATCH, COMPAREMODE

CURRENTLIMIT/CL

Purpose:

To restrict the DAC output voltage to within a defined range.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
			●	●

Format:

```
CURRENTLIMIT[axes] = <expression> {,<expression> ...}
v = CURRENTLIMIT[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●		●		100	$0.0 \leq x \leq 100.0$

Description:

The **CURRENTLIMIT** keyword restricts the output voltage of the DAC output. This can be used to protect motors and drives from excessive current or demand speeds. A value is a percentage of the maximum DAC output (±10V).

Example:

```
CURRENTLIMIT[0,1,2] = 50,50,60
```

will set the current limit of axes 0 and 1 to 50% and axis 2 to 60%.

Using **CURRENTLIMIT** with velocity amplifiers will restrict the maximum velocity.

Restrictions:

CURRENTLIMIT is only applicable to axes configured as Servo axes.

See also:

DAC, KINTLIMIT

CURRENTLIMITAREA/CLA

Purpose:

Returns the current I^2T area.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
			●	

Dot Parameters:

Axis - Axis No.

Format:

```
v = CURRENTLIMITAREA[axis]
```

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
MintDrive	●			●		0	$0 \le x \le 100$

Description:

Returns the current I^2T area as a percentage of the maximum area.

The maximum area is a function of the Peak current, the Peak Continuous current and the cycle count.

See also:

CURRENTLIMIT, CURRENTLIMITCONT, CURRENTLIMITMODE

CURRENTLIMITCONT/CLC

Purpose:

Sets or reads the continuous current that can be demanded.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
			●	●

Format:

```
CURRENTLIMITCONT[axes] = <expression> {,<expression> ...}
v = CURRENTLIMITCONT[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
MintDrive	●	●		●		100	$0 \le x \le 100$

Description:

Sets or returns the continuous current demand as a percentage of the maximum DAC output.

See also:

CURRENTLIMIT, CURRENTLIMITAREA, CURRENTLIMITMODE

CURRENTLIMITMODE/CLM

Purpose:

To set and read the how the I^2T current limiting operates.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
			●	

Format:

```
CURRENTLIMITMODE[axes] = <expression> {,<expression> ...}
v = CURRENTLIMITMODE[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
MintDrive	●	●		●		0	$0 \le x \le 1$

Description:

The following modes exist:

- 0 - Ignore (I^2T is not enabled);
- 1 - Fold Back (Demand current is folded back to the continuous current when the I^2T limit is reached).

See also:

CURRENTLIMIT, CURRENTLIMITAREA, CURRENTLIMITTIME

CURRENTLIMITTIME/CLT

Purpose:

Sets or reads the I^2T time.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
			●	

Format:

```
CURRENTLIMITMODE[axes] = <expression> {,<expression> ...}
v = CURRENTLIMITMODE[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
MintDrive	●	●		●		1000	$0 \le x \le 30000$

Description:

Reads or sets the maximum time that the demand current can equal the peak current before folding back.

See also:

CURRENTLIMIT, CURRENTLIMITAREA, CURRENTLIMITMODE

CURRENTMEAS/CMS

Purpose:

Reads the measured current.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
			●	●

Dot Parameters:

None.

Format:

```
v = CURRENTMEAS
```

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●						$0 \le x \le$ Drive Rating

Description:

Reads the measured current from the drive in Amps.

Example:

The following will monitor the measured current response:

```
TORQUE = 10
LOOP : ? CURRENTMEAS : ENDL
```

DACMODE/DCM

Purpose:

To control the use of the DAC.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Dot Parameters:

Channel - DAC Channel

Format:

```
DACMODE.channel = <expression>
v = DACMODE.channel
```

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●					$0 \le x \le 16$

Description:

NextMove BX and NextMove PCI have four 14 bit DAC outputs, normally used for controlling axes. The binary range is ±8191 counts giving ±10v. A motors tuning parameters derived on NextMove PC would be different to the parameters on NextMove BX/PCI due to the larger binary count on the DAC (±2047 on PC). The **DACMODE** keyword allows 12 bit emulation, removing the need to retune motor parameters when changing controllers.

DACMODE accepts a bitmap as follows:

Bit	NextMove	MintDrive / SN51	Description
0	●		14-bit mode, allowing a binary range of ±8191
1			*Reserved*
2			*Reserved*
3			*Reserved*
4	●	●	Inverts the DAC

On NextMove, if bits 0 through 3 are not set then **DACMODE** assumes 12 bit emulation, allowing a binary range of ±2047.

Mode	PCI	BX	PC	MintDrive	SN51
12 bit	●	●	●	●	●
14 bit	●	●			
DAC invert	●	●	●	●	●

The **DACMODE** effects all modes using the DAC channel, servo demand output, **TORQUE** output and direct control with the **DAC** keyword.

The other modes gives a greater resolution and therefore allows for finer motor control.

Changing DACMODE will require affected servo axes to be re-tuned.

Example:

```
DACMODE[0, 1, 2] = 1;    : REM 14 bit mode on channels 0, 1 and 2
DACMODE[3] = 0           : REM 12 bit emulation on channel 3
```

See also:

DAC, KPROP, TORQUE

DACMONITORAXIS/DMA

Purpose:

To specify which axis to monitor during DAC monitoring.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
DACMONITORAXIS.channel = <expression>
v = DACMONITORAXIS.channel
```

Dot Parameters:

Channel - DAC Channel

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●					0 - 7
MintDrive	●	●				0	0

Description:

Certain system parameters, such as axis velocity, can be monitored externally by outputting the value through a DAC channel. This can be useful for tuning and debugging purposes. The **DACMONITORAXIS** keyword specifies which axis is to be used for monitoring through the DAC channel.

Any axis can be monitored and multiple axis parameters can be monitored through the use of multiple DAC channels.

Note that this keyword is channel dependant and not axis dependent. The DAC channels on a NextMove expansion board can be used if one is present.

Example:

```
DACMONITORAXIS.3 = 0   :  REM Monitor axis 0 on DAC channel 3
DACMONITORMODE.3 = 1   :  REM Send axis 0 velocity to the DAC.
```

See also:

DAC, DACMONITORGAIN, DACMONITORMODE

DACMONITORGAIN/DMG

Purpose:

To specify a multiplying factor for use during DAC monitoring.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
DACMONITORGAIN.channel = <expression>
v = DACMONITORGAIN.channel
```

Dot Parameters:

Channel - DAC Channel

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●				1.0	0.0 - 8388607.0
MintDrive	●	●				1	

Description:

When using DAC monitoring, the parameter being monitored is just sent to the DAC channel. If the value is small, for example, less than 10, this equates to a voltage in the order of 0.05V. The **DACMONITORGAIN** allows the value to be scaled to produce a larger voltage output.

Example:

```
DACMONITORAXIS.3 = 0   :  REM Monitor axis 0 on DAC channel 3
DACMONITORMODE.3 = 4   :  REM Send axis 0 free spaces to the DAC.
DACMONITORGAIN.3 = 100 :  REM Scale output by 100
```

By setting a gain of 100, each free space in the buffer would be represented by 0.5V.

See also:

DAC, DACMONITORAXIS, DACMONITORMODE

DACMONITORMODE/DMM

Purpose:

To specify which axis parameter to monitor during DAC monitoring.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
DACMONITORMODE.channel. = <expression>
v = DACMONITORMODE.channel
```

Dot Parameters:

Channel - DAC Channel

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●				0	0 - 4
MintDrive	●	●				0	0, 1, 2, 3, or 5

Description:

Certain system parameters, can be monitored externally by outputting the value through a DAC channel. The **DACMONITORAXIS** keyword specifies which axis is to be used for monitoring through the DAC channel. The **DACMONITORGAIN** keyword can be used to scale the size of the output voltage.

The possible parameters to monitor are:

Mode	Meaning	NM	Mint Drive
0	Turns DAC monitoring off	●	●
1	The axis measured velocity is output in counts / second	●	●
2	The axis demand velocity is output in counts / second	●	●
3	The axis following error is output in counts	●	●
4	The free spaces in the axis move buffer is output	●	●
5	The axis demanded DAC output		●

The DAC can output values in the range ±2047 which equates to ±10V.

Any axis can be monitored and multiple axis parameters can be monitored through the use of multiple DAC channels. Note that this keyword is channel dependant and not axis dependent. The DAC channels on a NextMove expansion board can be used if one is present.

The axis owning the DAC channel to be used for monitoring must be configured as off (see `CONFIG`).

Whilst DAC monitoring is turned on, a DAC value set with the **DAC** keyword will be overwritten.

Example:

```
CONFIG.3 = _cfOFF       :  REM Turn axis 3 off so its DAC can be used
DACMONITORAXIS.3 = 0    :  REM Monitor axis 0 on DAC channel 3
DACMONITORMODE.3 = 1    :  REM Send axis 0 velocity to the DAC.
DACMONITORGAIN.3 = 10   :  REM Scale output by ten.
```

For example, if the DAC voltage is -5V, this equates to -1024 DAC counts. If the DAC is monitoring measured velocity with a monitor gain of 10.0, then the axis has a velocity of 102.4 counts / second.

Controller Specifics:

NextMove BX and PCI use a 14 bit DAC which has a of ±8191 counts. In the example above, a DAC voltage of -5V equates to -4095 DAC counts. With a gain of 10 this would indicate a velocity of 409.5 counts / second.

See also:

DAC, DACMONITORAXIS, DACMONITORGAIN

DACOFFSET/DCO

Purpose:

Apply a voltage offset to a DAC channel.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
DACOFFSET.channel = <expression>
value = DACOFFSET.channel
```

Dot Parameters:

Channel – DAC Channel Number

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●				0	-100.0 ≤ x ≤ 100.0

Description:

The **DACOFFSET** keyword can used to apply a constant voltage offset to a DAC. This would be useful where a load needs to held against gravity or to overcome drift in a drive. The offset is specified as a percentage of output voltage.
The offset is added to any DAC command voltage. The offset therefore applies for servo axes and for open loop control. Reading the **DAC** keyword will return the current output voltage, including the offset.

Example:

```
DACOFFSET.0 = -10   : REM Set an offset of –1.0 volts
```

See Also:

DAC, DACRAMP, TORQUE

DACRAMP/DCR

Purpose:

Specify the number of milliseconds over which the maximum DAC output will be ramped to zero.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
DACRAMP.channel = <expression>
v = DACRAMP.channel
```

Dot Parameters:

Channel – DAC Channel Number

Attributes:

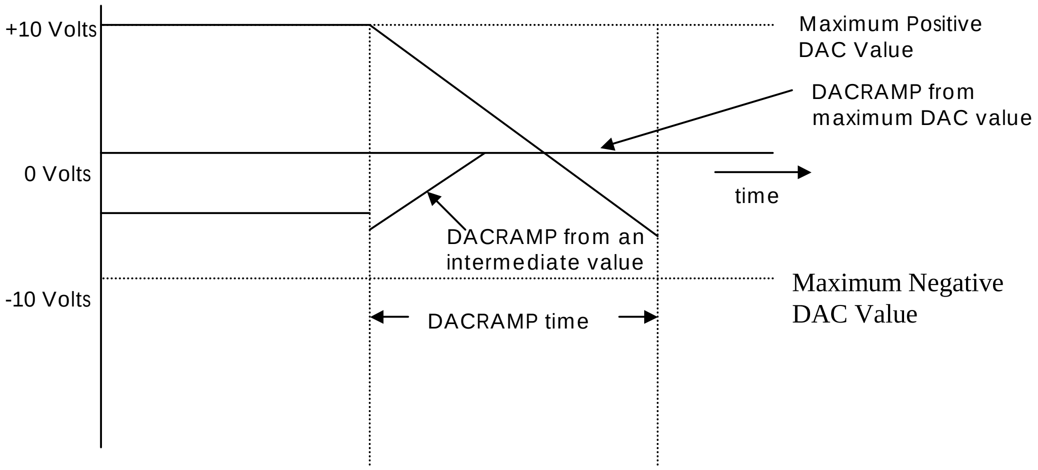
Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	0 ≤ x ≤ 8388607

Description:

Certain types of error conditions support the default action of DAC ramp (**_emDAC_RAMP**), these are:

- Following Errors
- Velocity Fatal Errors

If the default action to be taken in the event of these errors is setup to be ramp the DAC to zero volts (**_emDAC_RAMP**) then when an error occurs on an axis the DAC output is ramped to zero volts at the rate defined by **DACRAMP**.
The DACRAMP keyword defines how long (in milliseconds) that it would take to ramp the maximum DAC voltage to zero volts.



The above diagram illustrates the affect of **`DACRAMP`** on the DAC output.

Example:

```
FOLERRORMODE.0 = _emDAC_RAMP
DACRAMP.0 = 10
```

The above code will specify the default action to be taken in the event of a following error is to ramp the DAC to zero volts over the time period specified by **`DACRAMP`**.

See Also:

FOLERRORMODE, VELFATALMODE

DRPEVENT/DPR

Purpose:

Interrupt the host PC and generate an trappable event.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●				

Format:

```
DPREVENT = <expression>
```

Dot Parameters:

None

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	$0 \leq x \leq 255$

Description:

The **`DPREVENT`** keyword allows the user to interrupt the host passing an 8 bit value with the interrupt. This interrupt can be trapped by a host application using the **`installDPREventHandler`** function (see Mint v4 PC Programming Guide for further details on writing PC applications). The DPR Event is subject to the same priority scheme as all the other events. The priority of these events is:

Event	Priority
Error	Highest
CAN Open	
Baldor CAN	
Stop Switch	
Fast Position Latch	
Timer	

Event	Priority
Digital Input	
Comms	
DPR	
Move Buffer Low	
Axis Idle	Lowest

This means that a DPR Event may be held pending if a higher priority event is active.

Example:

```
DPREVENT = 10
```

This will generate an interrupt to the host PC which will call the user installable DPR Event handler passing a code of 10.

See Also:

EVENTACTIVE, EVENTDISABLE, EVENTPENDING

DPRFLOAT

Purpose:

Read and write a 32 bit floating point value to Dual Port Ram.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●			

Format:

```
DPRFLOAT (address) = <expression>
v = DPRFLOAT (address)
```

Parameters:

Address – Dual Port Ram location

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●					any floating point value

Description:

The Dual Port Ram is an area of Ram accessible by both the host PC and the controller, the size of this depends on the controller. The **DPRFLOAT** keyword allows the user to read and write 32 bit floating point values to any location within Dual Port Ram.

NextMove PCI

There is 4k of 32 bit wide Dual Port Ram on NextMove PCI, detail of the Dual Port Ram map can be found in then Mint v4 PC Programming Guide. The valid address range on NextMove PCI is 0x0000 to 0x0FFD inclusive.

NextMove PC

There is 1k of 16 bit wide Dual Port Ram on NextMove PCI, detail of the Dual Port Ram map can be found in then Mint v4 PC Programming Guide. The valid address range on NextMove PC is 0x0 to 0x3FD inclusive.

As the Dual Port Ram on NextMove PC is only 16 bits wide a 32 bit floating point value will take up 2 dual port ram locations.

See Also:

DPRWORD, DPRLONG

DPRLONG

Purpose:

Read and write a 32 bit integer value to Dual Port Ram

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●			

Format:

```
DPRLONG (address) = <expression>
v = DPRLONG (address)
```

Parameters:

Address – Dual Port Ram location

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●				0	any 32 bit integer value

Description:

The Dual Port Ram is an area of Ram accessible by both the host PC and the controller, the size of this depends on the controller. The **DPRFLOAT** keyword allows the user to read and write 32 bit integer values to any location within Dual Port Ram.

NextMove PCI

There is 4k of 32 bit wide Dual Port Ram on NextMove PCI, detail of the Dual Port Ram map can be found in then Mint v4 PC Programming Guide. The valid address range on NextMove PCI is 0x0000 to 0x0FFD inclusive.

NextMove PC

There is 1k of 16 bit wide Dual Port Ram on NextMove PCI, detail of the Dual Port Ram map can be found in then Mint v4 PC Programming Guide. The valid address range on NextMove PC is 0x0 to 0x3FD inclusive.

As the Dual Port Ram on NextMove PC is only 16 bits wide a 32 bit integer value will take up 2 dual port ram locations.

See Also:

DPRWORD, DPRFLOAT

DPRWORD

Purpose:

Read and write a 16 bit integer value to Dual Port Ram

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●			

Format:

```
DPRWORD (address) = <expression>
v = DPRWORD (address)
```

Parameters:

Address – Dual Port Ram location

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●				0	any 16 bit integer value

Description:

The Dual Port Ram is an area of Ram accessible by both the host PC and the controller, the size of this depends on the controller. The **DPRFLOAT** keyword allows the user to read and write 16 bit integer values to any location within Dual Port Ram.

NextMove PCI

There is 4k of 32 bit wide Dual Port Ram on NextMove PCI, detail of the Dual Port Ram map can be found in then Mint v4 PC Programming Guide. The valid address range on NextMove PCI is 0x0000 to 0x0FFD inclusive.

Care should be taken when writing 16 bit integer values to the dual port ram on NextMove PCI. The 16 bit value will not be sign extended and hence if it is not read back as a 16 bit value the perceived value will be different.

NextMove PC

There is 1k of 16 bit wide Dual Port Ram on NextMove PCI, detail of the Dual Port Ram map can be found in then Mint v4 PC Programming Guide. The valid address range on NextMove PC is 0x0 to 0x3FD inclusive.

See Also:

DPRLONG, DPRFLOAT

DRIVEPARAM/DRP

Purpose:

To get/set the value of the specified drive parameter.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
			●	●

Format:

```
V = DRIVEPARAM.param
DRIVEPARAM.param = <expression>
```

Dot Parameters:

Param – drive parameter

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
MintDrive, ServoNode 51	●	●					

Description:

Drive parameters are MintDrive and ServoNode 51 drive stage parameters. The DRIVEPARAM keyword should not be used where an individual Mint keyword is available.

No.	Parameter name	Mint Keyword	Range	Factory Default	Units
8	ServoNode 51 drive type. 1 – brushed 7 – brushless		0 or 7	0	

No.	Parameter name	Mint Keyword	Range	Factory Default	Units
1601	Resolver Align	**MOTORALIGNMENT**	0 to 360	Calculated value	Degrees
1602	UVW Alignment	**MOTORPHASEOFFSET**	0 to 360	Calculated value	Degrees
1603	Speed Filter	**MOTORSPEEDFILTER**	0 to 7	Calculated value	
1604	Feedback Align	**MOTORDIRECTION**	0 – Forward 1 – Reverse	0	
1605	Current Prop Gain	**KIPROP**	0 to 1000	50	
1606	Current Int Gain	**KIINT**	0 to 400	150	Hz
1607	Speed Prop Gain	**KVPROP**	0 to 1000	10	
1608	Speed Int Gain	**KVINT**	0 to 9.99	1.00	Hz
1611	Motor Inductance (line to line)	**MOTORINDUCTANCE**	0 to 98.3	Motor Dependent	mH
2002	Max Output Speed	**MOTORSPEEDMAX**	0 to 22500	Rated Motor Speed	RPM
2003	Peak Current Limit	**MOTORPEAKCURRENT**	0 – Peak Rated Current	Peak Control rating	Amps
2501	Motor Rated Amps	**MOTORRATEDCURRENT**	0 to 999.9	Factory Set	Amps
2502	Motor rated Poles	**MOTORPOLES**	0 to 100	4	Poles
2503	Feedback Type	**MOTORFEEDBACK**	0 – Resolver 1 – UVW Encoder 2 – Z Encoder	0	
2504	Encoder Lines	**MOTORENCODERLINES**		1024	
2505	Resolver Speed	**MOTORRESOLVERSPEED**	0 to 10	1	

See Also:

DRIVEFAULT

EFFORT/EF

Purpose:

Reads the instantaneous demand to the power stage of the drive.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
				●

Format:

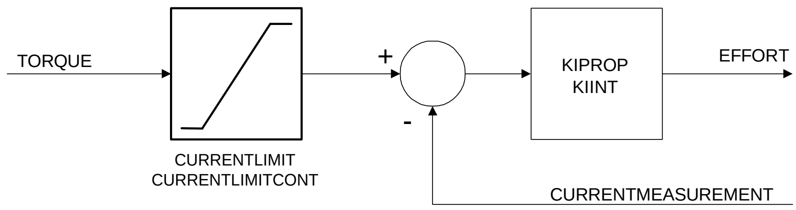
```
v = EFFORT
```

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
ServoNode 51	●					0	$-100 \leq x \leq 100$

Description:

Reads the instantaneous demand to the power stage. **EFFORT** is a percentage of the maximum demand that can be applied to the power stage. It can also be accessed via high speed capture.



See Also:

CURRENTMEAS, TORQUE

ENCODERMODE/ENM

Purpose:

To make miscellaneous changes to the Encoders.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●			●	●

Format:

```
ENCODERMODE.channel = <expression>
v = ENCODERMODE.channel
```

Dot Parameters:

Channel – Encoder Channel Number.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●				0	$0 \leq x \leq 7$
MintDrive & ServoNode 51	●	●				1	$0 \leq x \leq 7$

Description:

The keyword `Error! Reference source not found.` accepts a bitmap as follows:

Bit	Purpose	Description
0	Encoder direction	Can be used to swap the encoder A and B output channels, to overcome possible wiring problems. 0 – Normal 1 – Reversed
1	Encoder type	Can be used to select the encoder type: 0 – Differential line receiver 1 – Single-ended encoder
2	Input source	Selects the input source: 0 – Encoder inputs A and B 1 – Step (A) and Direction (B)

Example:

Check the polarity of the encoders:

```
CONFIG = 1
DRIVEENABLE = 1
TORQUE = 10
LOOP : PRINT VEL : ENDL
```

If the axis speed is negative, then it is likely that the encoders have been incorrectly wired. Confirm this by applying a negative **TORQUE** to see if the speed is positive.

To reverse the direction of the encoder outputs:

```
CONFIG = 1
ENCODERMODE = ENCODERMODE OR 1
```

```
DRIVEENABLE = 1
TORQUE = 10
LOOP : PRINT VEL : ENDL
```

Repeating the torque test should now give the correct polarity.

Note: Changing ENCODERMODE can affect position related keywords

Restrictions:

Must be set after the **CONFIG** keyword has been assigned, and again after **DEFAULT** or **LOOPTIME** has been used.

See Also:

ENCODER

ENCODERZACTIVELEVEL/ENL

Purpose:

Specify the active level of the encoder z pulse.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
	●	●		

Format:

```
ENCODERZACTIVELEVEL = <expression>
v = ENCODERZACTIVELEVEL
```

Dot Parameters:

None.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●				1	$0 \le x \le 1$

Description:

The keyword **ENCODERZACTIVELEVEL** configures the active state of the encoders Z pulse. Valid options are:

- 0 – Encoder Z pulse is considered active when it is low.
- 1 – Encoder Z pulse is considered active when it is high.

This encoder Z active level is global to all encoders and is not channel specific.

Example:

```
ENCODERZACTIVELEVEL = 1
```

Sets the active level of the all encoder Z pulses to be 1.

See Also:

HOME

ENCODERZLATCH/ENZ

Purpose:

To get and reset the state of an axis Encoder Z Latch.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
ENCODERZLATCH.channel = <expression>
v = ENCODERZLATCH.channel
```

Dot Parameters:

Channel – Encoder Channel Number.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●						0 or 1

Description:

The encoder ASIC which is fitted to the NextMove PCI, MintDrive and ServoNode controllers will latch the edge of an encoder Z pulse within hardware. Generally the width of an encoder Z pulse is too narrow to poll for reliably using software alone. When an encoders Z pulse is seen the occurrence of this is latched within the hardware. It is this latched value that the **ENCODERZLATCH** keyword reads. The action of reading this register clears it, hence when using the **ENCODERZLATCH** keyword to search for a Z pulse the value of **ENCODERZLATCH** must be read first to ensure that any existing hardware latch is cleared.

Example:

```
a = ENCODERZLATCH.0             : REM read and clear the encoder z latch
REPEAT                          : REM wait until the z pulse is seen
  REM do nothing
UNTIL ENCODERZLATCH.0 = 1
ENCODER.0 = 0                   : REM zero the encoder position
```

The above code example will firstly clear the encoder z latch, then wait until another z pulse occurs, at which point the position of the encoder is set to zero.

See Also:

ENCODER

ERRORINPUTMODE/EIM

Purpose:

Controls the default action taken in the event of an external error input interrupt.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
ERRORINPUTMODE[axes] = <expression> {, <expression>, …}
v = ERRORINPUTMODE[axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●		●		1	$0 \le x \le 6$

Description:

Sets the default action to be taken when an external error signal is detected.

The following modes are available.

Mode	Action
0	Ignore. A motion error is not generated if an error input is detected. The **AXISSTATUS** keyword will indicate the abort condition.
1	Performs a crash stop on the axis and deactivates the drive enable output and drops the enable relay.
2	Performs a crash stop on the axis. The drive and enable relay are left enabled.
3	Performs a controlled stop on the axis. The axis will decelerate at the **ERRORDECEL** rate. The drive and enable relay are left enabled.
4	Not Applicable
5	The error handler is called.
6	Not Applicable

See also:

ERRORINPUT, #ONERROR, ERRORSWITCH

ERRORMASK

Purpose:

Prevent specific errors conditions from calling the error handler.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
ERRORMASK [axes] = <expression> {, <expression>, …}
v = ERRORMASK[axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		134335	$0 \le x \le 134335$

Description:

The keyword **ERRORMASK** allows specific errors to be prevented from generating calls to the error handler. The default action for the error will still be performed but the call to the error handler will be prevented.

ERRORMASK accepts a bit patterns defined below:

Bit	Meaning	Mint Constant
0	Software abort	**_erABORT**
1	Forward hardware limit active	**_erFWD_HARD_LIMIT**
2	Reverse hardware limit active	**_erREV_HARD_LIMIT**
3	Forward software limit reached	**_erFWD_SOFT_LIMIT**
4	Reverse software limit reached	**_erREV_SOFT_LIMIT**
5	Following error limit exceeded	**_erFOLLOWING_ERROR**
6	*Reserved*	
7	Error input active	**_erERROR_INPUT**
8	*Reserved*	
9	Internal Drive Fault	**_erDRIVE_FAULT**
10	Analogue limit exceeded	**_erADC_ERROR**
11	Master/Slave synchronization error	**_erSLAVE_SYNC_ERROR**
12	*Reserved*	
13	Drive enable is inactive	**_erDRIVE_ENABLE_INACTIVE**
14	*Reserved*	

Bit	Meaning	Mint Constant
15	*Reserved*	
16	Internal drive watchdog	**_erDRIVE_WATCHDOG**
17	Velocity error limit has been exceeded	**_erVEL_FATAL**

If the bit is set then the error handler may be called for that error type.

Example:

```
ERRORMASK.2 = _erABORT + _erFOLLOWING_ERROR + _erERROR_INPUT
```

The above code will prevent all errors generated for axis 2 except software abort (**_erABORT**), following errors, (**_erFOLLOWING_ERROR**), and error input (**_erERROR_INPUT**) from calling the error handler.

See Also:

ABORTERRORMODE, ADCERRORMODE, AXISERROR, ERRORINPUTMODE, FOLERRORMODE, LIMITMODE, SOFTLIMITMODE, VELFATALMODE

EVENTACTIVE/EA

Purpose:

Indicates whether an event handler is currently active.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
v = EVENTACTIVE
```

Dot Parameters:

None

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●			

Description:

Indicates whether an event handler is active. The value returned is a bit pattern. If a bit is set then the corresponding handler is active.

Mint events are processed on a priority scheme indicated by the table below.

Priority	Bit	Event	Mint Subroutine
0: Highest	0	Serial Event (NextMove PCI only)	N/A
1	1	Error	**#ONERROR**
2	2	CAN Open	**#CAN1**
3	3	Baldor CAN	**#CAN2**
4	4	Stop switch	**#STOP**
5	5	Fast position latch	**#FASTIN or #FASTINx**
6	6	Timer	**#TIMER**
7	7	Digital input active	**#INx**
8: Lowest	8	Comms location changed by host PC	**#COMMSx**

See Also:

EVENTDISABLE, EVENTPENDING

FASTAUXENABLE/FAB

Purpose:

Manually clears the Aux. Encoders fast position latch.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
FASTAUXENABLE.channel = <expression>
```

Dot Parameters:

Channel – Auxiliary encoder channel.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All		●					0

Description:

Manually clears the hardware latch for the encoders fast position input. This is used when the **FASTAUXLATCHMODE** keyword is set to 2 so that further fast latches are disabled until re-enabled by the user.

The state of the latch can be read using **FASTAUXLATCH**. This indicates if an axis has had a position latch occur. Reading **FASTAUXLATCH** will clear the software status of the latch but this does not re-enable the hardware latch. Writing to **FASTAUXENABLE** also clears the software latch.

To clear the latch, a value of 0 must be written.

See also:

FASTAUXENCODER, FASTAUXLATCH, FASTAUXLATCHMODE, FASTAUXSELECT

FASTAUXLATCH/FAL

Purpose:

To read the Auxiliary Encoder fast interrupt latch.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
v = FASTAUXLATCH.channel
```

Dot Parameters:

Channel – Auxiliary encoder channel.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●					0	0 or 1

Description:

The **FASTAUXLATCH** keyword is used to determine if an axis has had a fast position latch occur since the last time that **FASTAUXLATCH** was read. Reading this keyword also clears the software latch.

The mode of fast latching is set with the **FASTAUXLATCHMODE** keyword. If mode 2 is used, then the **FASTAUXENABLE** keyword must be used to clear the hardware latch.

Where the generic **#FASTIN** handler is used with a fast latch mode 1, the hardware fast interrupt latch will only be reset automatically once the **FASTAUXLATCH** keyword has been read. This is to prevent latches from being lost to the user.

Example:

```
FASTAUXLATCHMODE = 2
FASTAUXSELECT = 2
...
#FASTIN2
?"Fast Auxiliary encoder = ", FASTAUXENCODER
WAIT = 1000
FASTAUXLATCH = 2
RETURN
```

See also:

FASTAUXENABLE, FASTAUXLATCHMODE

FASTAUXLATCHMODE/FAM

Purpose:

Sets the default action to be taken to clear the Aux. Encoders fast position latch.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
FASTAUXLATCHMODE.channel = <expression>
v = FASTAUXLATCHMODE.channel
```

Dot Parameters:

Channel – Auxiliary encoder channel

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●				0	$0 \le x \le 2$

Description:

Controls the method of clearing the Auxiliary encoders fast position latch.

Mode	Description
0	Always re-capture position. Using this mode the data within the fast position register will be updated on every activating edge on the appropriate fast interrupt line. The latch is cleared automatically.
1	Do not re-capture position until after the handler has completed. Using this mode the data in the fast position register will remain unchanged whilst in the user installed handler. When this handler exits the latch will automatically be cleared and enable further latching. If a users handler has not been defined then the latch will be processed as soon as the interrupt has been serviced.
2	Do not re-capture position until the latch has been cleared manually. Using this mode the register will remain unchanged until the user clears the latch manually using **FASTAUXENABLE**.
3	Disables the fast position latch for that auxiliary channel.

See also:

FASTAUXENABLE, FASTAUXENCODER, FASTAUXLATCH, FASTAUXSELECT.

FASTAUXSELECT/FAS

Purpose:

Maps the Auxiliary Fast position capture to a specific fast input.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
FASTAUXSELECT.channel = <expression>
v = FASTAUXSELECT.channel
```

Dot Parameters:

Channel – Auxiliary encoder channel

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●				2	

Description:

The **FASTAUXSELECT** function allows you to select which of the fast position capture inputs will capture an auxiliary encoder channel.

MintDrive, ServoNode 51

There are two capture inputs, 0 and 2. Use the **FASTAUXSELECT** keyword to select which input will cause the auxiliary encoder value to be captured. E.g.

```
REM Input 2 captures the auxiliary encoder.
FASTAUXSELECT.0 = 2
```

NextMove PCI

There are 4 capture inputs, 0,1,2 and 3. An expansion card adds 4 inputs, 20, 21, 22 and 23 and an auxiliary encoder. An encoder can only be captured using inputs on the same card. Use the **FASTAUXSELECT** keyword to select which input will cause the auxiliary encoder value to be captured. E.g.

```
REM Input 0 captures auxiliary encoder 0 and input 22 captures auxiliary
encoder 1.
FASTAUXSELECT.0 = 0
FASTAUXSELECT.1 = 22
```

NextMove BX

There are 4 capture inputs, 0,1,2 and 3. Use the **FASTAUXSELECT** keyword to select which input will cause the auxiliary encoder value to be captured. E.g.

```
REM Input 3 captures the auxiliary encoder.
FASTAUXSELECT.0 = 3
```

See also:

FASTAUXENABLE, FASTAUXENCODER, FASTAUXLATCH, FASTAUXLATCHMODE.

FASTENABLE/FEB

Purpose:

Manually clears the Encoders fast position latch.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
FASTENABLE[axes] = <expression> {, <expression>, …}
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All		●		●			0

Description:

Manually clears the hardware latch for the encoders fast position input. This is used when the **FASTLATCHMODE** keyword is set to 2 so that further fast latches are disabled until re-enabled by the user.

The state of the latch can be read using **FASTLATCH**. This indicates if an axis has had a position latch occur. Reading **FASTLATCH** will clear the software status of the latch but this does not re-enable the hardware latch. Writing to **FASTENABLE** also clears the software latch.

To clear the latch, a value of 0 must be written.

See also:

FASTENCODER, FASTLATCH, FASTLATCHMODE, FASTSELECT.

FASTLATCH/FLT

Purpose:

To read the axis fast interrupt latch.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
v = FASTLATCH[axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●			●		0	0 or 1

Description:

The **FASTLATCH** keyword is used to determine if an axis has had a fast position latch occur since the last time that **FASTLATCH** was read. Reading this keyword also clears the software latch.

The mode of fast latching is set with the **FASTLATCHMODE** keyword. If mode 2 is used, then the **FASTENABLE** keyword must be used to clear the hardware latch.

Where the generic **#FASTIN** handler is used with a fast latch mode 1, the hardware fast interrupt latch will only be reset automatically once the **FASTLATCH** keyword has been read. This is to prevent latches from being lost to the user.

Example:

```
#FASTIN0
  IF FASTLATCH.1 THEN latchPos = FASTPOS.1 : FASTENABLE.1 = 0
RETURN
```

See also:

#FASTIN, FASTPOS, FASTENABLE, FASTLATCHMODE

FASTLATCHMODE/FTM

Purpose:

Sets the default action to be taken to clear the Encoders fast position latch.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
FASTLATCHMODE[axes] = <expression> {, <expression>, …}
v = FASTLATCHMODE[axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●		●		0	$0 \le x \le 2$

Description:

Controls the method of clearing the encoders fast position latch.

Mode	Description
0	always re-capture position. Using this mode the data within the fast position register will be updated on every activating edge on the appropriate fast interrupt line. The latch is cleared automatically.
1	do not re-capture position until after the handler has completed. Using this mode the data in the fast position register will remain unchanged whilst in the user installed handler. When this handler exits the latch will automatically be cleared and enable further latching. If a users handler has not been defined then the latch will be processed as soon as the interrupt has been serviced.
2	do not re-capture position until the latch has been cleared manually. Using this mode the register will remain unchanged until the user clears the latch manually using **FASTENABLE**.
3	Disables the fast position latch for that axis.

See also:

FASTENABLE, FASTENCODER, FASTLATCH, FASTSELECT.

FASTSELECT/FS

Purpose:

Maps the Fast position capture to a specific fast input.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
FASTSELECT[axes] = <expression> {, <expression>, …}
v = FASTSELECT[axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●		●		0	

Description:

The **FASTSELECT** function allows you to select which of the fast position capture inputs will capture an axis position.

MintDrive, ServoNode 51

There are two capture inputs, inputs 0 and 2. Use the **FASTSELECT** keyword to select which input will cause the axis position to be captured. E.g.

```
REM Input 2 captures axis position.
FASTSELECT.0 = 2
```

NextMove PCI

There are 4 capture inputs, 0,1,2 and 3. An expansion card adds 4 inputs, 20, 21, 22 and 23. An axis can only be captured using inputs on the same card. Use the **FASTSELECT** keyword to select which input will cause an axis position to be captured. E.g.

```
REM Input 0 captures position on axis 0
REM Input 0 captures position on axis 1
REM Input 1 captures position on axis 2
REM Input 1 captures position on axis 3
FASTSELECT[0,1,2,3] = 0,0,1,1
```

NextMove BX

There are 4 capture inputs, 0,1,2 and 3. Use the **FASTSELECT** keyword to select which input will cause an axis position to be captured. E.g.

```
REM Input 0 captures position on axes 0, 1 and 2
REM Input 2 captures position on axis 3
FASTSELECT[0,1,2,3] = 0,0,0,2
```

See also:

FASTENABLE, FASTENCODER, FASTLATCH, FASTLATCHMODE.

FEEDRATE/FR

Purpose:

To set the slew speed of an individual move loaded in the move buffer.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
FEEDRATE[axes] = <expression> {,<expression> ...}
v = FEEDRATE[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●	●	40000, 1000	0.0 ≤ x ≤ 8388607.0

Description:

For a positional move, the motion profile is specified by the acceleration (**ACCEL**), deceleration (**DECEL**) and the slew speed. The **SPEED** and **FEEDRATE** keywords operate identically; both set the slew speed for a positional move. The way in which the slew speed is interpreted is controlled with the **FEEDRATEMODE** keyword.

In 'speed mode' (as set with the **FEEDRATEMODE** keyword), the **FEEDRATE** is applied to all positional moves, including those already in the buffer. In 'feedrate mode' the **FEEDRATE** only applies to new moves loaded into the move buffer. Moves already in the move buffer will be executed at the slew speed they were loaded with. The **FEEDRATE** keyword is useful in making Mint programs easier to read. When in 'feedrate mode', the **FEEDRATE** keyword is used to set slew speed. When in 'speed mode' the **SPEED** keyword is used to set slew speed.

> **Note that there is no difference between the SPEED and FEEDRATE keywords. Their functionality is determined by the FEEDRATEMODE keyword.**

Reading **FEEDRATE** returns the last value written. It does not return the slew speed of the currently executing move. See **FEEDRATEMODE** for further details.

FEEDRATE is applicable with the following move types:

- **MOVEA**/**MOVER**
- **VECTORA**/**VECTORR**
- **CIRCLEA**/**CIRCLER**
- **OFFSET**
- **HOLDTOANALOG**
- **INCA**/**INCR**

The slew speed, set with **FEEDRATE**, may not be achieved if the move length is not sufficient to achieve a trapezoidal profile.

If the **FEEDRATE** is changed whilst in 'speed mode', it will affect all moves loaded into the buffer that were in 'speed mode'. The move in progress will ramp using the current **ACCEL** and **DECEL** values to the new **SPEED**.

Slew speed can also be affected by the **FEEDRATEOVERRIDE** keyword which allows the **FEEDRATE** value to be modified by a percentage multiplier. See **FEEDRATEOVERRIDE** for further details.

Example:

```
FEEDRATE = 20  : REM Set slew speed to 20 user units/second
SPEED = 20     : REM Set slew speed to 20 user units/second
```

Both of these are equivalent.

```
FEEDRATEMODE = 1  : REM Set 'feedrate mode'
FEEDRATE = 50     : REM Slew speed of 50 uu/s
MOVER = 20
FEEDRATE = 30     : REM Set slew speed to 30 uu/s
MOVER = 20
GO
```

In the above example, the first move is executed with a slew speed of 50 user units per second and the second move is executed with a slew speed of 30 user units per second. Changing the slew speed to 30 does not affect the first move because the **FEEDRATEMODE** keyword was used to set the move buffer to 'feedrate mode'.

See also:

ACCEL, DECEL, FEEDRATEMODE, FEEDRATEOVERRIDE, MAXSPEED, SPEED

FEEDRATEMODE/FRM

Purpose:

To control the use of slew speed and feedrate override.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
FEEDRATEMODE[axes] = <expression> {,<expression> ...}
v = FEEDRATE[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	$0 \le x \le 3$

Description:

The **FEEDRATEMODE** keyword controls the how the slew speed set with **SPEED** or **FEEDRATE** is interpreted and also allows the affect of the **FEEDRATEOVERRIDE** keyword to be enabled or disabled.

FEEDRATEMODE takes a bit pattern as follows:

Bit	Meaning
0	Set to enable 'feedrate mode'. 'speed mode' otherwise.
1	Set to enable **FEEDRATEOVERRIDE**. Disabled otherwise.

In 'speed mode', bit 0 not set, slew speed for positional moves is set with the **SPEED** or **FEEDRATE** keywords. Any change in slew speed takes immediate effect on moves already loaded into the move buffer, the axis changing to the new slew speed at the current **ACCEL** or **DECEL** rate.

Example:

```
FEEDRATEMODE = 0   : REM speed mode
SPEED = 30         : REM slew speed 30 uu/s
MOVER = 50         : REM relative move of 50 units
GO                 : REM start the move
PAUSE OFFSET < 25  : REM wait until half way
FEEDRATE = 10      : REM Change slew speed to 10 uu/s
```

In this example, the move is started and ramps to a slew speed of 30 user units per second. When there is less than 25 units left to travel, the slew speed is changed to 10 user units per second. This will have an immediate affect with the axis decelerating to the new slew speed.

In 'feedrate mode', bit 0 set, slew speed for positional moves is set with the **SPEED** or **FEEDRATE** keywords. The current slew speed is loaded with each positional move. When a move is executed from the buffer, it uses the slew speed it was loaded with, regardless of the current slew speed.

The feedrate mode is stored as each move is loaded, therefore a move in the move buffer knows if it was loaded in 'speed' or 'feedrate' mode. When loading moves with feedrate mode enabled, S-ramping is automatically disabled. It is not possible to perform moves with S-ramping whilst in feedrate mode.

Feedrate mode is useful for programming changes in slew speed across a number of moves. This is useful in contouring applications where the slew speed of corners and complex shapes needs to be slower than that of longer smoother shapes.

When contouring is turned on, a move will be performed at the specified **FEEDRATE** or slower, if the **FEEDRATE** of the next loaded move dictates that the axis needs to decelerate. If the next **FEEDRATE** is higher than the current **FEEDRATE**, then the axis will not accelerate until it passes into that next move. I.E. Any acceleration or deceleration will always occur in the fastest of a pair of moves.

Example:

```
FEEDRATEMODE = 1   : REM feedrate mode
SPEED = 30         : REM slew speed 30 uu/s
MOVER = 50         : REM relative move of 50 units
GO                 : REM start the move
PAUSE OFFSET < 25  : REM wait until half way
FEEDRATE = 10      : REM Change slew speed to 10 uu/s
MOVER = 20 : GO    : REM move 20 units
```

In this example, the move is started and ramps to a slew speed of 30 user units per second. When there is less than 25 units left to travel, the slew speed is changed to 10 user units per second. This has no effect on the move in progress however. When the next move of 20 units starts, a slew speed of 10 user units per second is used.

Bit 1 controls the use of **FEEDRATEOVERRIDE**. If bit 1 is set, then the slew speed is modified by the percentage set with the **FEEDRATEOVERRIDE** keyword. This is regardless of bit 0, i.e. speed or feedrate mode. If bit 1 is not set, then the **FEEDRATEOVERRIDE** keyword is ignored and the move will execute at the desired slew speed.

Example:

```
FEEDRATEMODE = 2       : REM override enabled. Speed mode
SPEED = 20             : REM slew speed of 20 uu/s
FEEDRATEOVERRIDE = 50  : REM 50% override
MOVEA = 20 : GO       : REM Absolute move to position 20
PAUSE OFFSET < 20     : REM Wait until 20 units left to travel
FEEDRATEMODE = 0      : REM override disabled. Speed mode
```

In this example, a move to position 20 is started. The slew speed of this move is 10 since the override is set to 50%. When there is less than 20 units left to travel, **FEEDRATEOVERRIDE** is disabled so the axis will accelerate up to its full slew speed of 20 user units per second for the remainder of the move length.

See also:

ACCEL, DECEL, FEEDRATE, FEEDRATEOVERRIDE, SPEED

FEEDRATEOVERRIDE/FRO

Purpose:

Overrides the current speed or feedrate being used.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
FEEDRATEOVERRIDE[axes] = <expression> {,<expression> ...}
v = FEEDRATEOVERRIDE[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●	●	100	$0.0 \le x \le 8388607.0$

Description:

For a positional move, the motion profile slew speed can be defined by the **SPEED** or **FEEDRATE** keywords. If the **FEEDRATEMODE** keyword is used to set an axis to ‘feedrate mode’, slew speeds are loaded with each move and it is not possible to change the slew speed of a move by writing to the **FEEDRATE** or **SPEED** keywords. The **FEEDRATEOVERRIDE** keyword allows the slew speed of moves to be changed by defining an override percentage. If the **FEEDRATEOVERRIDE** is 100% then slew speed will be the value defined by **FEEDRATE** or **SPEED**. If the percentage is higher, the move will run faster and vice-versa.

FEEDRATEOVERRIDE can be turned on and off with the **FEEDRATEMODE** keyword.
FEEDRATEOVERRIDE applies to the following move types:

- **MOVEA**/**MOVER**
- **VECTORA**/**VECTORR**
- **CIRCLEA**/**CIRCLER**
- **OFFSET**

Example:

```
FEEDRATEMODE = 3      : REM feedrate mode, override enabled
FEEDRATE = 20         : REM Set feedrate to 20 user units/second
MOVER = 10            : REM Load a move of 10
GO
PAUSE IDLE
FEEDRATEOVERRIDE = 50 : REM 50% override
MOVER = 10
GO
```

In the above example, the first move is executed with a **FEEDRATE** of 20 user units/second. The second is executed at 10 user units/second because the **FEEDRATEOVERRIDE** is called with 50%.

See also:

ACCEL, DECEL, FEEDRATE, FEEDRATEMODE, SPEED

FIRMWARERELEASE/FWR

Purpose:

To read the release number of the firmware.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
v = FIRMWARERELEASE
```

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●						

Description:

Reads the release number of the firmware present on the Controller.

This number is unique to every version of firmware released.

The firmware release number is also displayed when the **VER** command is used.

Example:

```
PRINT FIRMWARERELEASE
```

The above example will display the release number of the Mint firmware on the terminal display.

See also:

VER

FOLERRORMODE/FEM

Purpose:

To determine the action taken on the axis in the event of a following error.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
FOLERRORMODE[axes] = <expression> {,<expression> ...}
v = FOLERRORMODE[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		1	$0 \le x \le 6$
MintDrive & ServoNode 51	●	●				1	$0 \le x \le 5$

Description:

Specifies the default action to be taken in the event of a following error motion error. A following error occurs when an axis exceeds its following error limit. **AXISSTATUS** can be used to check when this event occurs. If the **FOLERRORMODE** is set to check for the error then a motion error (**_erFOLLOWING_ERROR**) will be generated.

The maximum following error limit is set with the **FOLERRORFATAL** keyword and the following error can be read with the **FOLERROR** keyword.

FOLERRORMODE also affects the use of **IDLEPOS** when determining the result of the **IDLE** keyword.

Following error is only applicable on servo axes or for Stepper axes with encoders. The keyword takes one of the following values.

Mode	Description
0	Ignore. The following error limits are detected and the appropriate bit in **AXISSTATUS** is set, but no other action is taken in response to a following error limit being exceeded
1	The axis is crashed stopped. The axis enables including the relay output are immediately dropped.
2	Not Applicable
3	Not Applicable
4	Not Applicable
5	The error handler is called
6	Ramp the DAC to zero with a rate set with **DACRAMP**. The axis will be disabled when the DAC reaches zero and the error handler called. (NextMove only)

A call to **RESET** will clear a following error. Alternatively, the **FOLERRORMODE** can be set to ignore, 0, and **CANCEL** used to clear the error.

See also:

CANCEL, ERR, AXISERROR, FOLERROR, FOLERRORFATAL, RESET

FOLERRORWARNING/FEW

Purpose:

Sets the following error threshold before an axis warning is generated.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
FOLERRORWARNING[axes] = <expression> {,<expression> ...}
v = FOLERRORWARNING[axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●	●	1000	0.0 < x ≤ 8388607.0

Description:

In order to detect that an axis is beginning to have a problem, a following error warning limit is to be implemented. This allows the user to take corrective action before the fatal limit is reached. The warning value is set with the keyword **FOLERRORWARNING**.

The warning detection is performed in the servo loop at the same time as the fatal following error check. If the warning value is reached, the error handler will be called if it has been installed.

If a following error warning occurs the **ERR** value is set to 502 and bit 0 of **AXISWARNING** is set and the error handler is called.

The limit value is specified in user units.

The warning detection can be disabled by setting bit 0 of **AXISWARNINGDISBALE** to zero.

See Also:

AXISWARNING, AXISWARNINGDISABLE

FREQ/FQ

Purpose:

To set a constant frequency output.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
	●			

Format:

```
FREQ[axes] = <expression> {,<expression> ...}
v = FREQ[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●			-8000000 ≤ x ≤ 8000000

Description:

Allow user to set frequency output in the range of 150Hz to 8MHz. The axis must be configured off in order to control the frequency output directly. The direction of the direction output signal is controlled by the sign of the value passed to the keyword. A value that is under 150Hz will not produce any output on the pulse output. **FREQ** cannot be used if any of the stepper channels are configured as **_cfPWM**. The pulse and direction outputs can also be controlled directly with the **STEPPERIO** keyword.

Example:

```
CONFIG[4] = _cfOFF
FREQ[4] = 1000
```

will give a 1000Hz frequency output on stepper axis 4.

See also:

CONFIG, DAC, STEPDIRECTION

GEARING/GR

Purpose:

To set the percentage size for gearing compensation.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
GEARING[axes] = <expression> {,<expression> ...}
v = GEARING[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		100.0	-8388607.0 ≤ x ≤ 8388607.0
MintDrive & ServoNode 51	●	●				100.0	-20000 ≤ x ≤ 20000

Description:

All sampled master / slave systems have an inherent lag in the system. This lag is speed dependant, the faster the master axis, the larger the lag seen. Gearing compensation can be used to overcome this lag for the **FOLLOW**, **FLY** and **CAM** move types.

The **GEARING** keyword allows the lag or lead position of the slave axis to be controlled. The default value is 100% which is 0 lag. A value of 200% will make the slave axis lead the master axis by a value that is dependent on the slave axis speed. A value of -100% will increase the size of the lag on the slave. The gearing compensation is based on axis speed, so the actual size of any requested lead or lag will change at different speeds.

Example:

```
MASTERSOURCE.0 = 0     : REM Master is axis position
MASTERCHANNEL.0 = 3    : REM Master is axis 3
GEARING.0 = 100.0      : REM Compensate lag
GEARINGMODE.0 = 1      : REM Turn on gearing compensation
FOLLOW.0 = 1.0         : REM Start following the master
```

See Also:

CAM, FLY, FOLLOW, GEARINGMODE, MASTERCHANNEL, MASTERSOURCE

GEARINGMODE/GRM

Purpose:

To turn gearing compensation on.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
GEARINGMODE[axes] = <expression> {,<expression> ...}
v = GEARINGMODE[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●		●		0	0 or 1

Description:

All sampled master / slave systems have an inherent lag in the system. This lag is speed dependant, the faster the master axis, the larger the lag seen. Gearing compensation can be used to overcome this lag for the **FOLLOW**, **FLY** and **CAM** move types.

The percentage compensation size is set with the **GEARING** keyword.

`GEARINGMODE = 1` turns gearing compensation on.

`GEARINGMODE = 0` turns gearing compensation off.

Example:

```
MASTERSOURCE.0 = 0     : REM Master is axis position
MASTERCHANNEL.0 = 3    : REM Master is axis 3
GEARING.0 = 100.0      : REM Compensate lag
GEARINGMODE.0 = 1      : REM Turn on gearing compensation
FOLLOW.0 = 1.0         : REM Start following the master
```

See Also:

CAM, FLY, FOLLOW, GEARING, MASTERCHANNEL, MASTERSOURCE

HOMEPOS/HPS

Purpose:

To read the axis position at the completion of the datum cycle.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
v = HOMEPOS[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●			●	●		$-8388607.0 \le x \le 8388607.0$

Description:

Axis position is measured in user units from the last datum point. When an axis homes, the position is automatically set to zero. The **HOMEPOS** keyword returns the axis position as it was prior to it be zeroed. This is useful for checking for slip or lost steps.

Example:

```
HOME.0 = 1
PAUSE IDLE.0
POS.0 = HOMEPOS.0
```

This homes axis 0 on a switch and the encoder Z pulse. Axis position is then restored as if the position wasn't zeroed by the home routine.

See Also:

HOME, POS

HTA

Purpose:

Starts the hold to analog mode of motion in which the position of an axis is controlled in order to keep the value of an analog input channel at the specified value.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
HTA[axes] = <expression> {,<expression> ...}
v = HTA[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	$-100 \le x \le 100$

Description:

Hold To Analog (HTA) is a mode of motion which allows the value seen on one of the analog input channels to be held at a constant specified value where the position of the axis has a direct affect on the value seen on the analog channel. The **HTA** keyword specifies the desired analog hold value.

The axis position is controlled to minimize the error between the desired hold value and the measured analog value. The Hold To Analog algorithm uses a proportional gain (**HTAGAIN**) and a damping term (**HTADAMPING**) to control motor position. The normal PIDVF loop is also used to control the axis position.

The axis must be idle and free from errors before the move can be started. Only further HTA moves can be loaded on the axis to change to current hold value. The move type is subject to all synchronous and asynchronous error conditions. If an asynchronous error dictates that the **ERRORDECEL** must be used then the move will stop using **ERRORDECEL**.

The motion is subject to all NextMove error types.

The motion of the axis is at all times within the profile defined with the **ACCEL**, **DECEL** and **SPEED**/**FEEDRATE** keywords. The response of the axis to an error on the analog channel can therefore be easily controlled

Reading the keyword returns the current value being held to. If the axis is not currently in **HTA** mode, then reading the keyword will return the last hold value.

The analog channel to use is defined with the **HTACHANNEL** keyword. It also possible to set up an analog filter with the **HTAFILTER** keyword and to set up an error deadband with the **HTADEADBAND** keyword.

The motion can be terminated by the **STOP** keyword which will cause the axis to ramp down at the current deceleration rate or **CANCEL** which will cause the axis to crash stop.

The value passed to the keyword is the desired value to be seen on the analog input channel. The valid range of the hold value depends on the current configuration of the analog input channel. The analog input channel is set with the **HTACHANNEL** keyword and the analog channel configuration is set with the **ADCMODE** keyword.

The **GO** keyword is required to start motion. Hold To Analog can be used on servo or stepper axes.

Example:

```
AXES[2]             : REM Use axis 2
ADCMODE.6 = 2       : REM Configure channel 6 to be single ended, bipolar
```

```
HTACHANNEL = 6          : REM Use analog channel 6
HTADEADBAND = 2         : REM Ignore errors of 2 or under
HTAFILTER = 0.2         : REM Set a low analog filter value
HTAGAIN = 0.03          : REM set the gain term
HTADAMPING = 0.2        : REM Set the damping term
ACCEL = 200             : REM Set maximum accel rate
DECEL = 300             : REM Set maximum decel rate
SPEED = 20              : REM Set maximum move speed
HOLDTOANALOG = 25       : REM set the hold value to 25%
GO                      : REM Start the move
```

The above example sets up a sample **HTA** system and starts the move to hold at a value of 25.

See also:

ACCEL, ADCMODE, CANCEL, FEEDRATE, HTACHANNEL, HTADAMPING, HTADEADBAND, HTAFILTER, HTAKINT, HTAKPROP, SPEED, STOP

HTACHANNEL/HAC

Purpose:

Specifies the analog channel to use while in Hold To Analog mode for a particular axis.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
HTACHANNEL[axes] = <expression> {,<expression> ...}
v = HTACHANNEL[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●			$0 \le x \le 7$

Description:

The **HTACHANNEL** keyword specifies the analog channel to monitor during a Hold To Analog move on that axis. The analog channel can be in any configuration (**ADCMODE**) as long as it is not turned off the analog channel can have any configuration. The channel number must be a valid channel number. Reading the **HTACHANNEL** keyword returns the analog channel currently being used.

Example:

```
HTACHANNEL.6 = 7
```

Use analog channel 7 for a HTA move on axis 6.

The default channel assignments are as follows:

Axis	0	1	2	3	4	5	6	7
Analog Channel	0	1	2	3	4	5	6	7

See also:

ACMODE, HTA

HTADAMPING/HAD

Purpose:

Specifies the damping term used in the Hold To Analog algorithm.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
HTADAMPING[axes] = <expression> {,<expression> ...}
v = HTADAMPING[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	-8388607.0 ≤ x ≤ 8388607.0

Description:

The **HTADAMPING** keyword is used to set the damping term in the Hold To Analog algorithm. The algorithm used to minimize error is:

$$Velocity = Error \times Gain + \Delta Error \times Damping + SumOfErrors \times IntegralGain$$

HTADAMPING sets the *Damping* term in the above equation. A negative number can be used to reverse the move direction of the Hold To Analog move. Under normal circumstances, it is assumed that if the current analog value is lower than the desired hold value, that a positive direction move is required. If a negative direction move is required in this situation then the **HTADAMPING** and **HTAKPROP** terms should both be negative numbers.

Example:

```
HTADAMPING = 10.0     : REM Set HTA damping on axis 0 to 10
```

See also:

HTA, HTAKINT, HTAKPROP

HTADEADBAND/HDB

Purpose:

Specifies the analog error deadband.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
HTADEADBAND[axes] = <expression> {,<expression> ...}
v = HTADEADBAND[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	0 ≤ x ≤ 100

Description:

In the Hold To Analog move, the error is calculated as the difference between the measured analog value and the specified hold value. A deadband can be set up so that the system will only try and reduce the error when it becomes larger than the size specified with the **HTADEADBAND** keyword.

The deadband size is in analog counts. The value is specified as a positive number and is the maximum size of the absolute analog error before the system will respond.

Example:

```
HTADEADBAND = 0.20   : REM Ignore an error of less than 0.20%
```

See also:

ADCMODE, HTA, HTACHANNEL

HTAFILTER/HAF

Purpose:

Sets the factor for the analog input filter.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
HTAFILTER[axes] = <expression> {,<expression> ...}
v = HTAFILTER[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	$0 \le x \le 1$

Description:

Whilst a Hold To Analog move is in progress, the analog channel being used can be filtered by a simple first order filter. This allows any high frequency noise to be removed from the incoming analog signal. The **HTAFILTER** keyword allows the rate of smoothing to be set between 0 and 1. A value of 0 turns the filter off. The closer the value to 1, the less effect each new analog sample has on the analog value being used for following. For example, if the factor is set to 0.9999 then it will take a number of seconds for the analog value seen in the Hold To Analog move to respond to an actual change on the analog input channel.

Example:

```
HTAFILTER = 0.9999   : REM Slow response to analog channel changes
```

The filter is applied to an axis, not a specific input channel. The filter will then be used on whichever channel is assigned with the **HTACHANNEL** keyword for that axis. The filtering only applies to the Hold To Analog move type and the filtered value is not reflected in the value read from the **ADC** keyword or from the value updated in dual port RAM at any time.

See also:

HTA, HTADEADBAND

HTAKINT/HKI

Purpose:

Specifies the integral gain term used in the Hold To Analog force loop.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
HTAKINT[axes] = <expression> {,<expression> ...}
v = HTAINT [axes]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	-8388607.0 ≤ x ≤ 8388607.0

Description:

The **`HTAKINT`** keyword is used to set the gain term in the Hold To Analogue algorithm. The algorithm used to minimize error is:

$$Velocity = Error \times Gain + \Delta Error \times Damping + SumOfErrors \times IntegralGain$$

`HTAKINT` sets the *IntegralGain* term in the above equation. A negative number can be used to reverse the move direction of the Hold To Analogue move. Under normal circumstances, it is assumed that if the current analog value is lower than the desired hold value, that a positive direction move is required. If the **`HTADAMPING`**, **`HTAKPROP`** and **`HTAKINT`** terms are of opposite signs, then this changes the behavior of the algorithm.

If an HTA deadband is used then the integrator is set to zero when the analogue value falls within the deadband. The integrator is also zeroed when the integral gain term is changed.

Example:

```
HTAKINT[2] = 0.01   : REM Set a HTA integral gain of 0.01
```

See also:

HTA, HTADAMPING, HTAKPROP

HTAKPROP/HKP

Purpose:

Specifies the proportional gain term used in the Hold To Analog force loop.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
HTAKPROP [axes] = <expression> {,<expression> ...}
v = HTAKPROP [axes]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	-8388607.0 ≤ x ≤ 8388607.0

Description:

The **`HTAKPROP`** keyword is used to set the gain term in the Hold To Analog algorithm. The algorithm used to minimize error is:

$$Velocity = Error \times Gain + \Delta Error \times Damping + SumOfErrors \times IntegralGain$$

`HTAKPROP` sets the *Gain* term in the above equation. A negative number can be used to reverse the move direction of the Hold To Analog move. Under normal circumstances, it is assumed that if the current analog value is lower than the desired hold value, that a positive direction move is required. If a negative direction move is required in this situation then the **`HTADAMPING`** and **`HTAKPROP`** terms should both be negative numbers.

Example:

```
HTAGAIN[2] = 1.5   : REM Set a HTA gain of 1.5
```

See also:

HTA, HTADAMPING, HTAKINT

ICMDISABLE

Purpose:

Disables the ICM interface to the controller.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
ICMDISABLE.channel = <expression>
value = ICMDISABLE.channel
```

Dot Parameters:

Channel – Channel No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	0 or 1

Description:

Allows the ICM interface to be disabled. In certain circumstances it may be desirable to disable the ICM interface. For example if a section of code must execute quickly and should not be interrupts by ICM calls.

A value of 1 will disable the ICM interface.

A value of 0 will re-enable the ICM interface.

Valid channels are:

Description	Mint Constant	Value
RS232 port	`_tmRS232`	1
RS485 port	`_tmRS485`	2
Dual Port RAM	`_tmDPR`	4

When the ICM interface is disabled, all ICM calls other than a call to ICMDisable will fail.

Example:

```
ICMDISABLE._tmDPR = 1
PAUSE POS.0 < 10.0
OUT.0 = 1
ICMDISABLE._tmDPR = 0
```

The above example will disable the ICM interface over Dual Port RAM while it waits for the position of axis zero to fall below 10, at which point digital output 0 will be turned on and the ICM interface over Dual Port RAM re-enabled.

IDLEPOS/IDP

Purpose:

Reads or sets the Idle Following error bound.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
IDLEPOS[axes] = <expression> {,<expression> ...}
v = IDLEPOS[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●	●	1000	$0.0 \le x \le 8388607.0$
MintDrive & ServoNode 51	●	●		●	●	1000	$0 \le x \le 16000$

Description:

The keyword **IDLEPOS** is used by the keyword **IDLE** in determining whether the axis is in motion or not.

If the absolute Following Error, **FOLERROR**, is not within the bounds specified by **IDLEPOS**, then **IDLE** will indicate that the axis is still in motion.

Restrictions:

IDLEPOS is not checked for if **FOLERRORMODE** has been set to ignore following errors.

Example:

```
AXES[3]
SCALE = 4000
IDLEPOS = 0.01
IDLEVEL = 1
MOVEA = 13.5:GO
PAUSE IDLE
```

The above example sets the idle positional window for axis 3 to be 0.01 user units.

See also:

FOLERROR, IDLE, IDLEVEL

IDLEVEL/IDV

Purpose:

Reads or sets the Idle velocity bound.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
IDLEVEL[axes] = <expression> {,<expression> ...}
v = IDLEVEL[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●	●	5000.0	$0.0 \le x \le 8388607.0$
MintDrive	●	●		●	●	0	see below

The range is affected by **LOOPTIME** as follows:

LOOPTIME	Range
500	0 ≤ x ≤ 250,000
1000	0 ≤ x ≤ 125,000
2000	0 ≤ x ≤ 62,500

Description:

The keyword **IDLEVEL** is used by the keyword **IDLE** in determining whether the axis is in motion or not.

If the measured velocity, **VEL**, is not within the bounds specified by **IDLEVEL**, then **IDLE** will indicate that the axis is still in motion.

Example:

```
AXES[3]
SCALE = 4000
IDLEPOS = 0.01
IDLEVEL = 1
MOVEA = 13.5:GO
PAUSE IDLE
```

The above example sets the idle velocity window for axis 3 to be 1 user unit / second.

See Also:

IDLE, IDLEPOS, VEL

INITERROR/IER

Purpose:

Reports any errors that were detected during start-up.

Controllers Supported:

MintDrive

Dot Parameters:

None.

Format:

```
v = INITERROR
```

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●					0	See table

Description:

As the controller powers-up it will perform a number of system checks and attempt to read configuration data from EEPROM. **INITERROR** can be used to determine whether any of these error conditions were detected at power-up.

If Mint is being used, then any **AUTO** command will be ignored, if any of these error conditions occur. A corresponding error message will be reported and control will be passed to the command line.

Bit	Constant	Description
0	**_iePRODUCTION_EEPROM_INVALID**	MME Production EEPROM area checksum did not match
1	**_ieUSER_EEPROM_INVALID**	MME User EEPROM area checksum did not match
2	**_ieCARRIER_EEPROM_INVALID**	Carrier EEPROM checksum did not match
3	**_iePOWER1_EEPROM_INVALID**	Power EEPROM checksum did not match
4	**_ieFAILED_TO_READ_MME_EEPROM**	Failed to read EEPROM

5	**_ieFAILED_TO_READ_CARRIER_EEPROM**	Failed to read EEPROM
6	**_ieFAILED_TO_READ_POWER1_EEPROM**	Failed to read EEPROM
7	**_ieDSP_FAILED_TO_INITIALISE**	DSP Failed to Initialise
8	**_ieDSP_LOST_USER_DATA**	DSP detected Lost User Data
9	**_ieFAILED_TO_CLEAR_ALL**	DSP failed to clear Lost User Data condition
10	**_ieFAILED_TO_SET_WATCHDOG**	Failed to set DSP Watchdog timeout
11	**_ieEEPROM_DEFAULTS_USED**	Controller reset condition detected

INPUTDEBOUNCE/IDB

Purpose:

To specify the number of samples that a digital input must be sampled for in order for the input to be seen.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
INPUTDEBOUNCE = <expression>
value = INPUTDEBOUNCE
```

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●				1	1 - 10

Description:

Digital input channels are sampled on a periodic basis. On NextMove, this is every millisecond. If an input is active in hardware at the point of sampling, then that input is seen and processed according to the rules of **INPUTMODE**, **INPUTACTIVELEVEL**, **INPUTPOSTRIGGER** and **INPUTNEGTRIGGER**.

The **INPUTDEBOUNCE** keyword specifies the number if samples an input must be active for in order to be seen and then processed. This is useful in ‘noisy’ environments where false inputs can be seen and where a contactor may ‘bounce’ as it changes state.

The default value is one, i.e. the input is not debounced. A value of two for example means that the input must be present of two consecutive samples.

If debouncing is not on (set to 1), then the **IN** and **INSTATE** keywords read the state of the inputs when called. If debouncing is turned on (greater than 1), then the **IN** and **INSTATE** keywords return the last sample values.

The **INPUTDEBOUNCE** keyword affects the current I/O bank as set with the **BANK** keyword.

Example

In order to avoid noise affecting the digital inputs, a valid input signal will have a minimum duration of 5 milliseconds. This equates to a debounce value of 6.

```
INPUTDEBOUNCE = 6
```

See also:

BANK, INPUTMODE

KACCEL/KA

Purpose:

To set the servo loop acceleration feed forward gain.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
KACCEL[axes] = <expression> {,<expression> ...}
v = KACCEL[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	$0 \le x \le 10000$
MintDrive	●	●		●		0	$0 \le x \le 255$

Description:

Sets the digital servo loop acceleration feed forward gain term. This term is designed to reduce velocity overshoots on high acceleration moves. Due to the quantization of the positional data and the speed of the servo loop, for the acceleration feed forward term to affect the servo loop the acceleration of the axis must exceed 1,000,000 encoder counts/s^2.

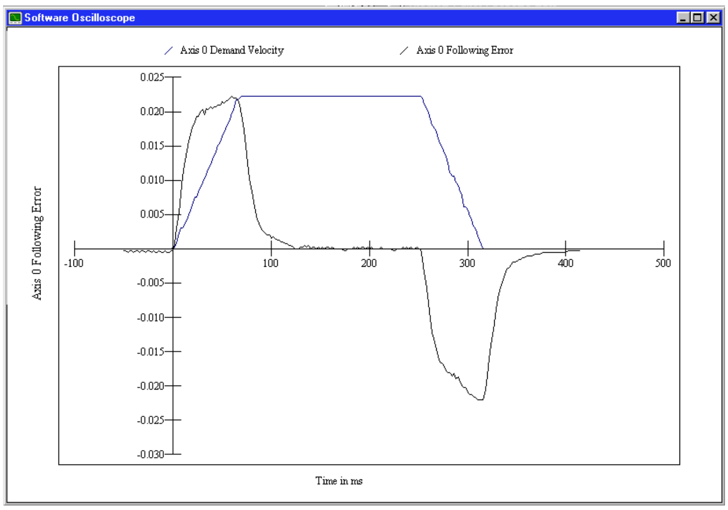


Figure 4: No acceleration feed forward.

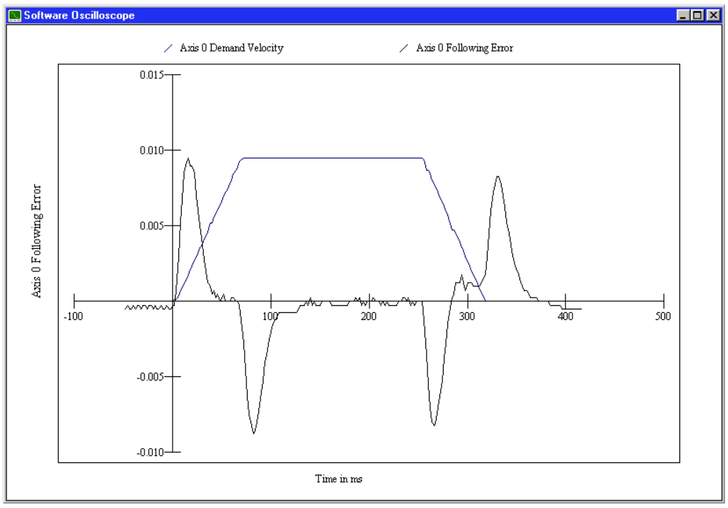


Figure 5: With acceleration feed forward.

Example:

```
KACCEL.0 = 45
```

Sets the acceleration feed forward term for axis 0 to be 45.

See also:

CURRLIMIT, DGAIN, KINT, KINTRANGE, KVEL, KVELFF

KEYS

Purpose:

To remap the layout of the keys on a Baldor CAN keypadnode.

Format:

```
KEYS "Up to 64 ASCII Characters"
```

The KEYS keyword allows you to configure your own keypad for any particular ASCII characters in an 8x8 array. The ASCII string following the keyword will define the keys.

If for example you have a keypad with 16 keys arranged in a 4x4 pattern, they might be mapped onto the 8x8 matrix as shown:

Assume you are using a 4x4 keypad and require the following layout:

9	8	7	A
6	5	4	B
3	2	1	<CR>
-	0	.	<BS>

where <CR> is carriage return, and <BS> is backspace or delete.

This is placed as follows in the 8x8 matrix:

9	8	7	A	.	.	.	.
6	5	4	B	.	.	.	.
3	2	1	<CR>	.	.	.	.
-	0	.	<BS>	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

The KEYS keyword will be defined as follows:

```
KEYS "987A....654B....321@....-0.~.................................."
REM   | row 1 | row 2 | row 3 | row 4 | row 5 | row 6 | row 7 | row 8 |
```

If keys are not used within the 8x8 matrix, they must be padded out with an arbitrary character. In this case a decimal dot is used.

Since <CR> and <BS> are not directly supported using a standard keyboard, KEYS will interpret '@' as carriage return and will return ASCII code 13. '~' is used to interpret back space and will return ASCII code 8.

The default KEYS string is:

```
.9.87FED\010W4.Z56-X\0150.VYU.AB1C.23 K.....OLN...P.IMTQ....JHR.....SG
```

If the KEYS string is specified with less than 64 characters, then those not specified are not remapped.

KINTMODE/KIM

Purpose:

To control when integral action will be applied in the servo loop.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
KINTLIMIT[axes] = <expression> {,<expression> ...}
v = KINTLIMIT[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	$0 \le x \le 3$
MintDrive & ServoNode 51	●	●				0	$0 \le x \le 2$

Description:

Sets or returns a software switch that controls how the servo loop integral term is applied during motion.

KINTMODE can take one of following values:

Value	Constant	Action
0	**_itNEVER**	The integral term is never applied (the integrator is grounded).
1	**_itALWAYS**	The integral term is always applied. **KINT** and **KINTLIMIT** control the effect of the integral term.
2	**_itSMART**	The integral term is only applied when the axis demand speed is constant ie: when the axis is at rest or moving at constant speed. At all other times the integrator is grounded.
3	**_itSTEADY_STATE**	The integral term is only applied when the axis demand speed is zero i.e: when the axis is at rest. At all other times the integrator is grounded.

Example:

```
KINT = 0.34
KINTLIMIT = 10
```

```
KINTMODE = 2
```

The integral action will have a maximum effect of 1.0V on the DAC output. This will only occur when the axis is at rest or is moving at a constant velocity. During acceleration or deceleration, the integral action is turned off.

DEFAULT can be used to set all servo loop gains to zero and **KINTLIMIT** to 100%.

Restrictions:

KINTMODE is only applicable on servo axes.

See also:

CURRENTLIMIT, DAC, DACLIMITMAX, DEFAULT, KDERIV, KPROP, KINT, KINTLIMIT, KVEL, KVELFF, TORQUE

LOOPTIME/LT

Purpose:

Sets the servo loop update rate.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
LOOPTIME = <expression>
v = LOOPTIME
```

Dot Parameters:

None.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove PCI	●	●				1000	200, 500 or 1000
NextMove	●	●				1000	500 or 1000
ServoNode 51	●	●				1000	1000, 2000
MintDrive	●	●				1000	500, 1000, 2000

Description:

The servo loop algorithm is applied to all the servo axes on a periodic basis. It calculates the required voltage to output to the axis DACs based on the current measured position and the required demand position.

The regularity and frequency of this algorithm affects the performance of the servo drives. Generally speaking, only very high performance servo drives actually require the high frequency loop closure. The response time of the physical system is often measured in milliseconds. The more often the loops are closed in a given period the less processor time is available for the application program code.

Example:

To set the servo loop time to 500 micro-seconds use:

```
LOOPTIME = 500
```

Restrictions:

To change the **LOOPTIME** all axis must be Disabled.

Controller Specifics:

On NextMove, the stepper axes are updated at the same frequency as the profiler. The profiler rate can be changed with the **PROFILETIME** keyword.

On MintDrive and ServoNode 51, changing **LOOPTIME** will set all motion variables, including **FASTLATCH**, to their power-up state.

See Also:

CONFIG, DEFAULT, PROFILETIME

MAXSPEED/MS

Purpose:

To set a limit for the speed demanded on an axis.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
MAXSPEED[axes] = <expression> {,<expression> ...}
v = MAXSPEED[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●	●	0	0 - 8388607.0
MintDrive	●	●		●	●	0	see below

On MintDrive the range depends upon the setting of **LOOPTIME** as follows:

LOOPTIME	Range
500	$0 \le x \le 8000000$
1000	$0 \le x \le 4000000$
2000	$0 \le x \le 2000000$

Description:

With the use of **FEEDRATEOVERRIDE** and performing interpolated moves at a given path speed, it is possible to demand velocities from an axis above the value set with the **SPEED** or **FEEDRATE** keywords. The **MAXSPEED** keyword allows a maximum speed to be set for an axis. This maximum speed will be used by the following move types:

- **CIRCLEA**/**CIRCLER**
- **FOLLOW** (velocity mode only)
- **HOME**
- **INCA**/**INCR**
- **JOG**
- **MOVEA**/**MOVER**
- **VECTORA**/**VECTORR**

The axis speed will be limited to approximately the maximum speed. If the maximum speed is reached, motion will continue and a warning will be indicted in the **AXISWARNING** keyword. When the demand speed falls below the maximum speed, this bit will be cleared.

In the case of multi axis moves, vectors and circles, only the maximum speed for master axis is used and this is taken to be the maximum path speed allowed. In the situation of axes having different maximum speeds, the slowest should be used for the master axis.

A maximum speed of 0.0 will turn off maximum speed checking.

Example:

```
MAXSPEED[0,1,2,3] = 30;
```

None of the servo axes can exceed a speed of 30 user units/second.

See Also:

AXISSTATUS, FEEDRATE, FEEDRATEOVERRIDE, SPEED

MISCERROR/MER

Purpose:

To read or clear the miscellaneous error flag.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
MISCERROR = <expression>
v = MISCERROR
```

Dot Parameters:

None.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●				0	see below

Description:

The keyword **MISCERROR** returns a bitmap indicating that the following miscellaneous errors have occurred:

Bit	Meaning	PCI	PC	BX	MD	SN51
0	12 volt supply rail down		●	●		
1	Output short circuit for bank 0	●			●	●
2	Output short circuit for bank 1	●				
3	Output short circuit for bank 2	●				
4	RS232 receive overrun			●	●	●
5	RS232 transmit overrun			●	●	●
6	RS232 receive framing error			●	●	●
7	CAN serial redirection overrun	●	●	●	●	●
8	Servo Tick overrun	●	●	●	●	●
9	RS485 receive error			●	●	●
10	RS485 transmit overrun			●	●	●
11	RS485 receive framing error			●	●	●
12	Output driver not powered or missing for bank 0	●				
13	Output driver not powered or missing for bank 1	●				
14	Output driver not powered or missing for bank 2	●				

Writing zero to **MISCERROR** will clear the error flag. See the MISCERRORDISABLE keyword for the predefined constants.

See also:

#ONERROR, AXISERROR, ERR, ERL, MISCERRORDISABLE

MISCERRORDISABLE/MED

Purpose:

To enable or disable miscellaneous errors calling the error handler.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
MISCERRORDISABLE = <expression>
v = MISCERRORDISABLE
```

Dot Parameters:

None.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●				32496	see below

Description:

MISCERRORDISABLE is a bitmap of the associated **MISCERROR** bits, which if set will prevent the error handler (**#ONERROR**) being called in response to that miscellaneous error being detected.

The error will still be flagged in **MISCERROR**.

Bit	Mint Constant	Meaning
0	**_me12V**	12 volt supply rail down
1	**_meOUTPUT_FAULT0**	Output short circuit for bank 0
2	**_meOUTPUT_FAULT1**	Output short circuit for bank 1
3	**_meOUTPUT_FAULT2**	Output short circuit for bank 2
4	**_meRS232_RECEIVE_OVERRUN**	RS232 receive overrun
5	**_meRS232_TRANSMIT_OVERRUN**	RS232 transmit overrun
6	**_meRS232_RECEIVE_ERROR**	RS232 receive framing error
7	**_meCAN_RECEIVE_OVERRUN**	CAN serial redirection overrun
8	**_meSERVO_TICK_OVERRUN**	Servo Tick overrun
9	**_meRS485_RECEIVE_OVERRUN**	RS485 receive error
10	**_meRS485_TRANSMIT_OVERRUN**	RS485 transmit overrun
11	**_meRS485_RECEIVE_ERROR**	RS485 receive framing error
12	**_meOUTPUT_POWER0**	Output driver not powered or missing for bank 0
13	**_meOUTPUT_POWER1**	Output driver not powered or missing for bank 1
14	**_meOUTPUT_POWER2**	Output driver not powered or missing for bank 2

The 12V Fail, Output short circuits and servo tick overrun cannot be disabled.

Setting a bit in **MISCERRORDISABLE** prevents that error from causing a call to **#ONERROR**. The following miscellaneous errors have a default action that takes place when the error occurs. The default action will always be performed, regardless of the state of the **MISCERRORDISABLE** keyword.

Bit 0: 12Vfail. All axes are disabled.
Bits 1,2 and 3: Output driver faults. All outputs are de-activated in that bank.
Bit 8: Servo tick overrun. The **LOOPTIME** and **PROFILETIME** keywords are reset to their default values.

See also:

#ONERROR, MISCERROR

MOVEBUFFERFREE/MBF

Purpose:

To return the number of free spaces in the move buffer for the specified axis.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
v = MOVEBUFFERFREE[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●						$0 \leq x \leq 100$

Description:

Each axis maintains a data structure, into which moves are loaded, known as the move buffer. The size of this buffer and hence the number of moves that can be loaded at any time can be configured with the **MOVEBUFFERSIZE** keyword.

If the buffer is full and an attempt is made to load a valid move type, program execution will wait until a free slot in the buffer becomes available. If the moves in the buffer require the **GO** keyword in order to start, the controller will hang in that state.

This situation can be avoided by using **MOVEBUFFERFREE** to return the number of free spaces left in the move buffer for the specified axis. If no moves are loaded then the number of free moves will be the buffer size. As moves are loaded, the number of free spaces will decrease to zero, indicating that the buffer is full.

The only exception to this is when following. The **FOLLOW** move type will appear to not occupy any space in the move buffer. Only one **FOLLOW** move can be loaded though so this does not present a problem.

Example:

```
LOOP
  IF MOVEBUFFERFREE > 1 THEN GOSUB LoadNextMove
  ..
ENDL
```

This loops round testing the number of spaces available in the buffer. If there are 2 or more free spaces then it calls a user routine to load another move.

See Also:

GO, MOVEBUFFERSIZE

MOVEBUFFERID/MBI

Purpose:

To attach or read back a 16 bit identifier from the move buffer.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
MOVEBUFFERID[axes] = <expression> {,<expression> ...}
v = MOVEBUFFERID[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	$1 \le x \le 65535$

Description:

When a move is loaded into the move buffer, it is possible to give that move an identifier with the **MOVEBUFFERID** keyword. Reading the **MOVEBUFFERID** keyword will return the identifier of the currently executing move in the buffer. This allows a number of moves to be loaded into the buffer in advance and still be able to keep track of which move is currently executing. The identifier of the previous move to be executed can be read with the **MOVEBUFFERIDLAST** keyword.

An identifier can be used with the following move types:

- **MOVEA**/**MOVER**
- **VECTORA**/**VECTORR**
- **CIRCLEA**/**CIRCLER**
- **FLY**

If there is no move loaded in the move buffer or the move is not one of the above types, then the identifier will read as zero.

Example:

```
MOVEBUFFERID = 1       : REM Identifier of 1
MOVER = 20
MOVEBUFFERID = 2       : REM Identifier of 2
MOVER = 45.5
MOVEBUFFERID = 3       : REM Identifier of 3
MOVER = -34.2
MOVEBUFFERID = 4       : REM Identifier of 4
MOVEA = 0 : GO
PAUSE MOVEBUFFERID = 3 : REM Wait until the identifier is 3
OUT1 = 1
```

The above program loads a number of moves, each with different identifiers. When the third move starts, output 1 will be turned on.

See also:

MOVEBUFFERFREE, MOVEBUFFERIDLAST, MOVEBUFFERSIZE, MOVEBUFFERSTATUS

MOVEBUFFERIDLAST/MBL

Purpose:

To read a 16 bit identifier from the move buffer.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
v = MOVEBUFFERIDLAST[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●					0	$1 \le x \le 65535$

Description:

When a move is loaded into the move buffer, it is possible to give that move an identifier with the **MOVEBUFFERID** keyword. Reading the **MOVEBUFFERID** keyword will return the identifier of the currently executing move in the buffer. When the axis is idle or performing an unsupported move type, the identifier is zero. The **MOVEBUFFERIDLAST** keyword allows the identifier of the previously executed move to read, even if no move is in progress.

An identifier can be used with the following move types:

- **MOVEA**/**MOVER**
- **VECTORA**/**VECTORR**
- **CIRCLEA**/**CIRCLER**
- **FLY**

Example:

```
MOVEBUFFERID = 1       : REM Identifier of 1
MOVER = 20
MOVEBUFFERID = 2       : REM Identifier of 2
MOVER = 45.5
MOVEBUFFERID = 3       : REM Identifier of 3
MOVER = -34.2
MOVEBUFFERID = 4       : REM Identifier of 4
MOVEA = 0 : GO
PAUSE MOVEBUFFERID = 3 : REM Wait until the identifier is 3
OUT1 = 1
PAUSE MOVEBUFFERIDLAST = 4 : REM Wait until all moves have finished
```

The above program loads a number of moves, each with different identifiers. When all the moves have finished, **MOVEBUFFERID** would give 0, **MOVEBUFFERIDLAST** would give 4.

See also:

MOVEBUFFERFREE, MOVEBUFFERID, MOVEBUFFERSTATUS

MOVEBUFFERLOW/MBW

Purpose:

To set or return the number of free spaces in the move buffer before a move buffer low event is generated.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
MOVEBUFFERLOW [axes] = <expression> {,<expression> ...}
v = MOVEBUFFERLOW [axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	$0 \le x \le 99$

Description:

When the number of free spaces in the move buffer exceeds the threshold set by **MOVEBUFFERLOW** an event will be generated. This event can be trapped by the user installed move buffer low handler which

is installed using the function **installMoveBufferLowHandler**, see the Mint v4 PC Programming Guide for further details on trapping events on the host.

Move buffer low events can only be trapped using an embedded or a host application, not within Mint.

Example:

```
AXES[2]
MOVEBUFFERSIZE = 10
MOVEBUFFERLOW = 3
```

The above example changes the **MOVEBUFFERSIZE** to 10. Setting **MOVEBUFFERLOW** to 3 means that when there are more than 3 free spaces in the move buffer a 'move buffer low' event will be generated.

See Also:

MOVEBUFFERSIZE

MOVEBUFFERSIZE/MB

Purpose:

To set or return the size of the move buffer allocated on the specified axis.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
MOVEBUFFERSIZE[axes] = <expression> {,<expression> ...}
v = MOVEBUFFERSIZE[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●		●		2	$2 \le x \le 100$

Description:

Each axis maintains a data structure into which moves are loaded, known as the move buffer. The size of this buffer and hence the number of moves that can be loaded at any time can be configured to suit an application. The size of the move buffer controls how many moves can be loaded without calling the **GO** keyword and also affects the efficiency of axis contouring. The maximum number of multi-axis moves that can be loaded is limited by the axis with the smallest move buffer.

If the move buffer on an axis is full and another move on that axis is set up, Mint will wait for a space in the buffer to become available before moving onto the next line of code. Care must taken however since it possible to fill the move buffer and have a move pending without reaching a **GO** statement to execute the moves. It is therefore recommended that a **GO** statement is issued after each move is set up. The **MOVEBUFFERFREE** keyword can be used to check the amount of available space in the buffer before loading a move.

Each axis has a default move buffer size of 2 moves (the minimum permitted). This can be extended up to 100 per axis (memory space permitting). In order to change the move buffer size, the axis must be idle.

Example:

```
CONFIG[5,6] = _cfOFF;
MOVEBUFFERSIZE[5,6] = 30;
CONFIG[5,6] = _cfSTEPPER;
```

This changes the move buffer size to 30 on axes 5 and 6.

See also:

CONFIG, GO, MOVEBUFFERFREE

MOVEBUFFERSTATUS/MBS

Purpose:

To return information about the move buffer.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
v = MOVEBUFFERSIZE[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●			●		8	$0 \le x \le 119$

Description:

It is possible that each move in the move buffer has been loaded with a slightly different way. The **MOVEBUFFERSTATUS** keyword allows information about the currently executing move to be read.

The status returned is a bit pattern as follows:

Bit	Description
0	1: the current move was loaded in 'feedrate mode'. 0: the current move was loaded in 'speed mode'
1	1: the current move has **FEEDRATEOVERRIDE** enabled 0: the current move has **FEEDRATEOVERRIDE** disabled
2	1: the current move has contouring turned on 0: the current move has contouring turned off
3	1: the move buffer is empty
4	1: the move buffer is full
5	1: the move in the buffer has been suspended (**SUSPEND**) and is at rest
6	1: the end of the current move is a stop point caused by inter-vector angle control

Bits 5 and 6 were added in Mint v4.2 Build 1213.

If there is no move currently loaded in the move buffer then the status value will be 8. Bits 3 and 4 are mutually exclusive.

Example:

```
status = MOVEBUFFERSTATUS[3]  : REM Read status of axis 3
if status & 3 = 1 DO          : REM If bit 0 set and bit 1 not set
  PRINT " Move is a fast traverse "
ENDIF
```

The above example will print a message if the current move in the buffer is in feedrate mode but has feedrate override disabled.

See also:

FEEDRATEMODE, FEEDRATEOVERRIDE, MOVEBUFFERFREE, MOVEBUFFERSIZE, MOVEBUFFERID

NUMBEROF/NO

Purpose:

To return information about the hardware available on the Controller.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
v = NUMBEROF.type
```

Dot Parameters:

Type - Hardware Type

Description:

The **NUMBEROF** keyword can be used to return information about what hardware is present. The type of hardware available is as follows:

Type value	Constant	Meaning	NM	MD	SN
0	**_noSERVOS**	the number of possible servo axis	●	●	●
1	**_noSTEPPERS**	the number of possible stepper axis	●	●	●
2	**_noINPUTS**	the number of digital inputs	●	●	●
3	**_noOUTPUTS**	the number of digital outputs	●	●	●
4	**_noDACS**	the number of DAC channels	●	●	●
5	**_noADCS**	the number of ADC channels	●	●	●
6	**_noAUXENCODERS**	the number of auxiliary encoder channels	●	●	●
7	**_noAXES**	the number of controllable axes	●	●	●
8	**_noRELAY**	the number of relays	●	●	●
9	**_noCAN**	the number of CAN channels	●	●	●
10	**_noAUXDAC**	the number of Auxiliary DAC channels	●	●	●
11	**_noSERIAL**	the number of serial channels	●	●	●
12	**_noAXIS_CHANNELS**	the highest addressable channel or axis number.	●		
13	**_noICM_FUNCS**	the number of ICM functions	●		
14	**_noOUTPUT_BANKS**	the number of digital output banks	●		
15	**_noINPUT_BANKS**	the number of digital input banks	●		

Example:

A configuration file is to written that will run on both NextMove PC and NextMove BX.

```
IF NUMBEROF._noSERVOS <> 4 THEN PRINT “Error.  Not enough servos” : END

IF NUMBEROF._noINPUTS < 24 DO
  REM Configure I/O for 24 inputs
  ..
ELSE
  REM Configure I/O for 16 inputs
  ..
ENDIF

IF NUMBEROF._noAXUENCODERS = 1 DO
  REM Auxiliary encoder available for product position verification
  ..
ENDIF
```

See Also:

VIEW

NVFLOAT/NVF

Purpose:

To write to non-volatile memory.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●				●

Format:

```
NVFLOAT ( location ) = <expression>
v = NVFLOAT ( location )
```

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove PCI, ServoNode 51	●	●					-8388607.0 ≤ x ≤ 8388607.0

Description:

The NextMove PCI and ServoNode 51 controllers have an area of non-volatile RAM available for user access. The **NVFLOAT** keyword allows the reading and writing of floating point values to this user area of non-volatile memory.

The data in this memory is retained when the card is powered down. The data stored is duplicated. This allows the controller to determine if a write was in progress when power was removed. The error **erNON_VOLATILE_MEMORY_ERROR** (37) will be returned if the data is corrupted.

The data must be read in the same format as it was written. Writing data to a location as a float and reading it back as a long will not give the same data. Within Mint, it is recommended that **NVFLOAT** is used as all data is floating point.

On NextMove PCI, there are 3072 addressable locations accessed from 0 to 3071 inclusive.

On ServoNode 51, there are 1024 addressable locations accessed from 0 to 1023 inclusive.

Example:

```
REM Store the number of products and product length in NVRam
NVFLOAT (0) = productLength
NVLONG (1) = productNumber
```

See Also:

NVLONG

NVLONG/NVL

Purpose:

To write to non-volatile memory.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●				●

Format:

```
NVLONG ( location ) = <expression>
v = NVLONG ( location )
```

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove PCI, ServoNode 51	●	●					-8388607 ≤ x ≤ 8388607

Description:

The NextMove PCI and ServoNode 51 controllers have an area of non-volatile RAM available for user access. The **NVFLONG** keyword allows the reading and writing of integer values to this user area of non-volatile memory.

The data in this memory is retained when the controller is powered down. The data stored is duplicated. This allows the controller to determine if a write was in progress when power was removed. The error **`erNON_VOLATILE_MEMORY_ERROR`** (37) will be returned if the data is corrupted.
The data must be read in the same format as it was written. Writing data to a location as a float and reading it back as a long will not give the same data. Within Mint, it is recommended that **`NVFLOAT`** is used as all data is floating point.
On NextMove PCI, there are 3072 addressable locations accessed from 0 to 3071 inclusive.
On ServoNode 51, there are 1024 addressable locations accessed from 0 to 1023 inclusive.

Example:

```
REM Store the number of products and product length in NVRam
NVFLOAT (0) = productLength
NVLONG (1) = productNumber
```

See Also:

NVFLOAT

OFFSETDISTANCE/OFD

Purpose:

Specifies the distance over which offset moves of mode 2 or 3 will occur.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
			●	●

Format:

```
OFFSETDISTANCE[axes] = <expression> {,<expression> ...}
v = OFFSETDISTANCE[axes]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
MintDrive & ServoNode 51	●	●		●	●	1000	$1.0 \le x \le 8388607.0$

Description:

When an offset of mode 2 or 3 is executed the offset distance is taken as the distance over which to perform the offset.

Example:

```
OFFSETDISTANCE.1 = 5
```

This sets the distance over which the offset will occur to be 5 user units.

See also:

OFFSET, OFFSETMODE, OFFSETSTATUS

OFFSETMODE/OFM

Purpose:

Define the mode of operation on the **`OFFSET`** keyword.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
			●	●

Format:

```
OFFSETMODE[axes] = <expression> {,<expression> ...}
v = OFFSETMODE[axes]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
MintDrive & ServoNode 51	●	●		●		0	$0 \le x \le 3$

Description:

The velocity profile of an **OFFSET** move is determined by the **OFFSETMODE** of the axis.

Mode	Predefined Constant	Action
0	**_ofSTANDARD**	Perform offset using **ACCEL** and **DECEL** defined for the axis with no regard for **MAXSPEED** or reversing.
1	**_ofSPEED_LIMIT**	Perform offset using **ACCEL** and **DECEL** defined for the axis but do not exceed maximum speed or change direction.
2	**_ofACCEL_MOD**	Modify the **ACCEL** and **DECEL** parameters of the axis in order to perform the offset over the specified distance without constant velocity.
3	**_ofSPEED_MOD**	Modify the **SPEED** parameter of the axis in order to perform the offset over the specified distance at a constant velocity.

OFFSETMODE 0 - default

In this mode the offset is performed using the **ACCEL** and **DECEL** as defined for the axis. The maximum speed of the offset is the base speed plus the speed defined by the **SPEED** keyword. This mode ignores any maximum speed defined for the axis and may change direction during an offset move.

OFFSETMODE 1 - SPEED AND DIRECTION CHECK

As with OFFSETMODE 0 this move uses the **ACCEL** and **DECEL** defined for the axis performing the offset. However in this mode the motor speed will not exceed that defined by the **MAXSPEED** keyword and the motor will not change direction during an offset move.

OFFSETMODE 2 - MODIFIED ACCEL AND DECEL

This mode inherits the speed restrictions of mode 1 but takes place over the distance defined by the keyword **OFFSETDISTANCE**. This mode modifies the **ACCEL** and **DECEL** parameters of the axis during the offset move to produce a triangular velocity profile. This mode is designed to be used when the axis is travelling at a constant base speed as the speed of the axis at the time of the execution of the offset is used to calculate the time in which to perform the offset. If the **ACCEL** and **DECEL** parameters required for the new profile exceed those defined for the axis the offset will abort and set **OFFSETSTATUS** accordingly.

OFFSETMODE 3 - MODIFIED SPEED

This mode inherits the speed restrictions of mode 1 and as with offset mode 2 takes place over the distance defined by the keyword **OFFSETDISTANCE**. This mode modifies the **SPEED** parameter in order to perform the offset at the minimum speed to achieve the offset required in the distance specified. It uses the current **ACCEL** and **DECEL** parameters for the axis. If the speed calculated for the new profile exceeds that defined for the axis the offset will abort and set **OFFSETSTATUS** accordingly.

Example:

```
MASTERSOURCE.1 = _msPOS
MASTERCHANNEL.1 = 0
```

```
FOLLOWMODE.1 = 0
FOLLOW.1 = 0
OFFSETMODE.1 = 0
OFFSET.1 = 20
GO.1
```

Controller Specifics:

On MintDrive **OFFSETMODE** = 2, the profiled Offset will be clipped at **MAXSPEED** if it exceeds it and thus the Offset would not be achieved in the desired **OFFSETDISTANCE**.

Restrictions:

Any changes in the master axis velocity will not be tracked by the **OFFSET** move, only the underlying **FOLLOW** motion. This can result in the offset not performing as expected.

See also:

OFFSET, OFFSETDISTANCE, OFFSETSTATUS

OFFSETSTATUS/OFS

Purpose:

Read the status of the previous offset move.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
			●	●

Format:

```
v = OFFSETSTATUS[axes]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●			●			$0 \leq x \leq 127$

Description:

OFFSETSTATUS is a bit pattern which represents the status of the previous OFFSET move.

Bit	Predefined Constant	Meaning
0	**_osMAXSPEED**	The maximum speed as defined by the **MAXSPEED** keyword has been reached during the **OFFSET** move.
1	**_osMINSPEED**	The axis stopped during the **OFFSET** move.
2	**_osMAXACCEL**	The **OFFSET** was aborted as the acceleration required to perform the desired profile exceeded that defined by the **ACCEL** keyword.
3	**_osMAXDECEL**	The **OFFSET** was aborted as the deceleration required to perform the desired profile exceeded that defined by the **DECEL** keyword.
4	**_osPROFILEERROR**	The **OFFSET** was aborted because the velocity profile required infinite acceleration.
5	**_osZEROACCEL**	The **OFFSET** was aborted because the velocity profile required zero or negative acceleration.

Example:

```
MASTERSOURCE.1 = _msPOS
MASTERCHANNEL.1 = 0
FOLLOWMODE.1 = 0
FOLLOW.1 = 0
OFFSETMODE.1 = 0
OFFSET.1 = 20
GO.1
```

```
IF OFFSETSTATUS.1 <> 0 THEN REM OFFSET failed
```

See also:

OFFSET, OFFSETMODE, OFFSETDISTANCE

PLATFORM/PTM

Purpose:

To return the platform type.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
PRINT PLATFORM
v = PLATFORM
```

Dot Parameters:

None.

Description:

PLATFORM returns the current platform type. The following is a list of platform types:

Value	Platform
-1	Invalid
0	*Reserved*
1	*Reserved*
2	NextMove BX
3	NextMove PC
4	*Reserved*
5	*Reserved*
6	*Reserved*
7	*Reserved*
8	*Reserved*
9	NextMove PCI
10	MintDrive
11	ServoNode 51

See Also:

NUMBEROF, VIEW

POSDEMAND/PSD

Purpose:

To return the instantaneous demand position.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Dot Parameters:

Axis - Axis No.

Format:

```
POSDEMAND[axes] = <expression> {,<expression> ...}
v = POSDEMAND[axis]
```

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●		●	●		-8388607 ≤ x ≤ 8388607

Description:

Returns the current demand position set by the profiler.

When a move is not being profiled by the controller (i.e. when the axis is Idle), it is possible to set a **`POSDEMAND`**. This will give a stepped response to the new demanded position.

The change in the demanded position from the current position cannot exceed the current value of **`FOLERRORFATAL`**.

See also:

MOVEA, MOVER, POS, POSTREMAINING, POSTARGET

PRECISIONINCREMENT/PCI

Purpose:

Sets the theoretical distance between each of the values in the compensation tables.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Mint Format:

```
PRECISIONINCREMENT[axes] = <expression> {,<expression> ..}
v = PRECISIONINCREMENT[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

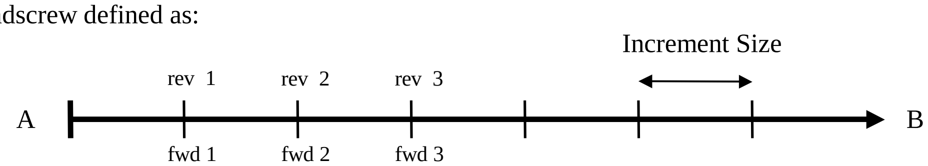
Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●	●	1.0	-8388607.0 ≤ x ≤ 8388607.0

Description:

The theoretical spacing between the values in the compensation tables is set with the **`PRECISIONINCREMENT`** keyword. If dual compensation tables are being used, the increment for the forward and reverse tables must be the same.

The sign of the increment relates to the direction of the leadscrew in relation to the axis orientation.

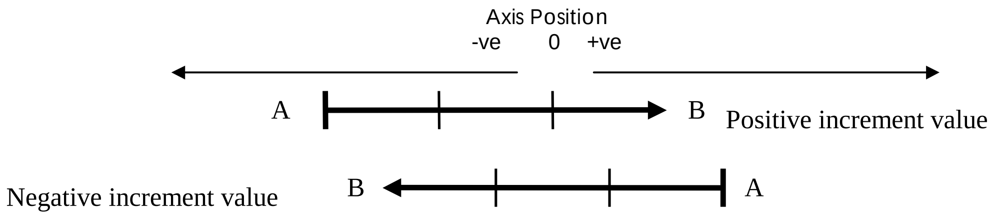
Given a leadscrew defined as:



where *fwd x* are the actual values for travel in direction from A to B and *rev x* are the actual values for travel in the direction from B to A. For single table compensation, the *fwd x* values would apply in both direction.

If the leadscrew was laid in the machine such that travel from A to B results in a +ve change in axis position, then the increment value is positive.

If the leadscrew was laid in the machine such that travel from A to B results in a -ve change in axis position, then the increment value is negative.



Example:

```
SCALE.0 = 1000              : REM Scale to mm
PRECISIONINCREMENT.0 = 10.0 : REM Distance between compensation values 10.0 mm
```

If the axis is scaled to mm, then the theoretical distance between the tables values is 10.0 mm.

See also:

PRECISIONMODE, PRECISIONOFFSET, PRECISIONTABLE, BACKLASHMODE

PRECISIONMODE/PCM

Purpose:

Controls the action of leadscrew compensation.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Mint Format:

```
PRECISIONMODE[axes] = <expression> {,<expression> ..}
v = PRECISIONMODE[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

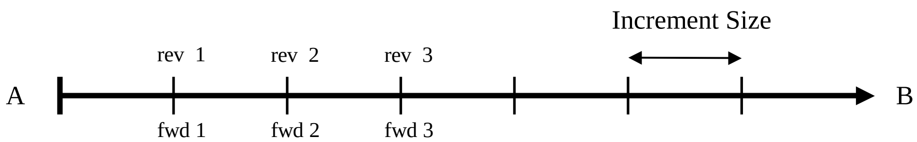
Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	0 - 2

Description:

Leadscrew compensation has two modes of operation, single table mode and dual table mode.

`_lsTABLE` A leadscrew compensation table is defined with the **`PRECISIONTABLE`** keyword. This single table is used to calculate position corrections, regardless of the direction of travel along the table. A precision table must be specified prior to turning the compensation on. Backlash compensation can be used in conjunction with single table mode to account for changes in direction.

`_lsDUAL_TABLE` Two leadscrew compensation tables are defined with the **`PRECISIONTABLE`** keyword. The forward table is used for travel along the table in a forward direction, the reverse table is used for travel in a reverse direction. Forward and reverse direction are based on how the table is specified. It is assumed that the tables relate to the leadscrew as:



where *fwd x* are the actual values for travel in direction from A to B and *rev x* are the actual values for travel in the direction from B to A. For single table compensation, the *fwd x* values would apply in both direction.

Backlash compensation must be turned off before dual table compensation can be used. A value of 0, **`_lsOFF`**, turns of leadscrew compensation.

Reading axis position with **`POS`**, will return the compensated axis position, i.e. the theoretical position along the leadscrew. Reading the axis Encoder value will return the uncompensated position, i.e. including any extra motion required for the compensation.

Mint Example:

```
BACKLASHMODE = _blOFF         : REM Turn off backlash compensation
PRECISIONMODE = _lsDUAL_TABLE : REM Turn on dual table compensation
```

See also:

PRECISIONINCREMENT, PRECISIONOFFSET, PRECISIONTABLE, BACKLASHMODE

PRECISIONOFFSET/PCO

Purpose:

Sets the distance between the start of the leadscrew and axis zero position.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Dot Parameters:

Axis - Axis No.

Mint Format:

```
PRECISIONOFFSET[axes] = <expression> {,<expression> ..}
v = PRECISIONOFFSET[axis]
```

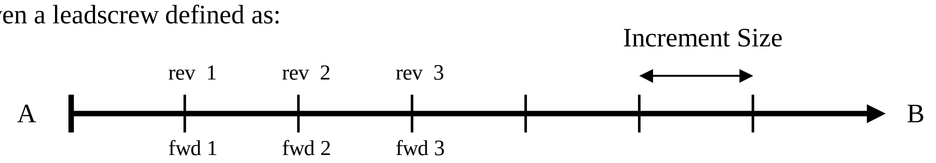
Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●	●	0.0	-8388607.0 - 8388607.0

Description:

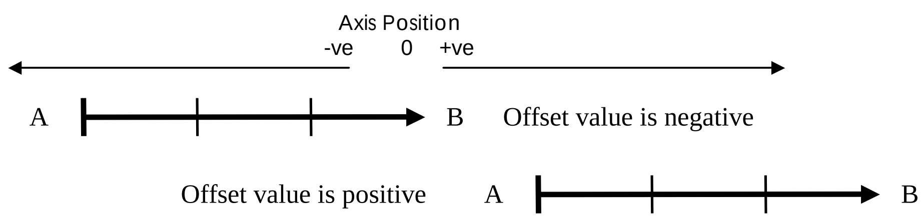
Axis zero position may not coincide with the start of the leadscrew. The **`PRECISIONOFFSET keyword`** allows this offset value to be specified.

Given a leadscrew defined as:



where *fwd x* are the actual values for travel in direction from A to B and *rev x* are the actual values for travel in the direction from B to A. For single table compensation, the *fwd x* values would apply in both direction.

The offset value is the position of point A relative to axis zero position.



Mint Example:

```
HOME.2 = 0                  : REM Home axis in a negative direction
PAUSE IDLE.2                : REM Wait for move to complete
POS.2 = -50                 : REM Set axis position to -50
PRECISIONOFFSET.2 = -50.0   : REM If axis is at point A of leadscrew then the
                              REM offset is -50.0
```

See also:

PRECISIONINCREMENT, PRECISIONMODE, PRECISIONTABLE, BACKLASHMODE

PRECISIONTABLE

Purpose:

Loads the leadscrew compensation tables.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Mint Format:

```
PRECISIONTABLE (axis, forwardTable, reverseTable)
```

Dot Parameters:

Axis – Axis No.
Forward Table – Array of data for the forward table
Reverse Table – Array of data for the reverse table

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove		●					

Description:

Leadscrew compensation allows for a precision table, supplied by the leadscrew manufacturer, to be given to Mint so that it will automatically compensate for positional inaccuracies along the length of the leadscrew.

The data is specified in an array where the values are the actual positions relative to the end of the leadscrew. The first element of the array specifies the number of elements in the precision table.

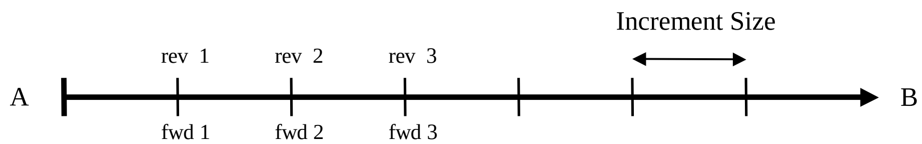
Mint Example:

```
DIM single(11) = 10, 10.04, 20.02, 30.01, 40.00, 49.99,
                     59.97, 70.00, 80.00, 90.01, 100.02
```

The array *single* contains the compensation data. There are 10 points in the table and the values are the measured positions along the leadscrew.

For dual table compensation, two tables would be defined, one for forward travel and one for reverse travel.

Given a leadscrew defined as:



where *fwd x* are the actual values for travel in direction from A to B and *rev x* are the actual values for travel in the direction from B to A. For single table compensation, the *fwd x* values would apply in both direction.

If only one table is being defined, then the reverse table is specified is NULL. Whether one or two tables are used depends on the mode set with **PRECISIONMODE**.

Example:

A axis has a leadscrew for which bi-directional data is available. The leadscrew starts at position 100 and is laid such that travel in the forward direction (from A to B) results in a negative change in axis position.

```
DIM forward(11) = 10, 1.01, 2.01, 3.00, 4.00, 5.01,
                      6.00, 6.99, 8.00, 9.01, 10.00
DIM reverse(11) = 10, 1.00, 2.01, 3.01, 4.01, 5.00,
                      6.00, 6.98, 7.99, 9.00, 10.00

PRECISIONTABLE (0, forward, reverse)
PRECISIONINCREMENT.0 = -1 : REM Increment size is 1 in a negative direction.
PRECISIONOFFSET.0 = 100   : REM Point A is at position 100
PRECISIONMODE.0 = _lsDUAL_TABLE : REM Dual table operation.
```

See also:

PRECISIONINCREMENT, PRECISIONMODE, PRECISIONOFFSET, BACKLASHMODE

PROFILETIME/PT

Purpose:

Sets the profiler update rate.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●				

Format:

```
PROFILETIME = <expression>
v = PROFILETIME
```

Dot Parameters:

None.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove PCI	●	●				2000	1000 or 2000

Description:

The profiler on NextMove is where the motion profiles are generated and the stepper axes are processed. By default, the profiler runs every 2ms. Increasing this rate will provide a smoother demand position profile to the servo loop but at the expense of using more CPU processing time. Generally speaking, only very high performance servo or stepper drives will benefit from a faster profiler rate. The more often the loops are closed in a given period the less processor time is available for the application program code.

Example:

To set the profile loop time to 1000 micro-seconds use:

```
PROFILETIME = 1000
```

Restrictions:

To change the **PROFILETIME,** all axis must be disabled.

See Also:

CONFIG, LOOPTIME

PWMONTIME/PWM

Purpose:

To control the duty cycle of a PWM output train.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
	●			

Format:

```
PWMONTIME[axes] = <expression> {,<expression> ...}
v = PWMONTIME[axis]
```

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		1000	-16000 to 16000

Description:

The four stepper axes can be configured as PWM or stepper outputs. Configuring one axis as PWM will change all axes currently configured as stepper into PWM. Once in PWM mode, the frequency and duty cycle of the pulse train can be controlled.

The **PWMONTIME** keyword controls the duty cycle or mark space ratio of the pulse train. It specifies the amount of time that the output is on (active) in 125 ns ticks. A negative value sets the stepper axis direction pin low.

Example:

```
PWMPERIOD = 4000
PWMONTIME = 1500
```

This would give a pulse train where the pulse output was active for 187.5 µs and inactive for 312.5µs.

PWM is not available on a stepper NextMove daughter board.

See also:

BOOST, CONFIG, FREQ, PULSE, PWMPERIOD, STEPDIRECTION

PWMPERIOD/PWP

Purpose:

To control the period of a PWM output train.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
	●			

Format:

```
PWMPERIOD = <expression>
v = PWMPERIOD
```

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●				2000	-1000 to 16000

Description:

The four stepper axes can be configured as PWM or stepper outputs. Configuring one axis as PWM will change all axes currently configured as stepper into PWM. Once in PWM mode, the frequency and duty cycle of the pulse train can be controlled.

The period is defined as the number of 125 ns ticks. The four PWM outputs must share the same PWM period. This will default to 250 microseconds (4KHz). The PWM period can be set or read back with the following functions:

Example:

```
PWMPERIOD = 8000
```

This would set a PWM period to 1ms (1KHz).

See also:

BOOST, CONFIG, FREQ, PULSE, PWMONTIME, STEPDIRECTION

SPLINE/SPL

Purpose:

Perform a SPLINE move.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
SPLINE[axis] = <expression> {,<expression> ...}
v = SPLINE[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●			$0 \le x \le 7$

Description:

A spline move provides a means of specifying motion where arbitrary position and velocity information needs to be specified in terms of time.

A spline move is created by breaking the motion into a number of segments. Each segment defines the how far the axis will travel in a given time duration and optionally the velocity of the axis at the end of that segment. Once these segments have been defined, NextMove will interpolate between them and 'fill in' the missing information, producing a smooth path as it goes. The user controls the size and accuracy of these segments. The more critical the axis motion, the more segments that can be defined.

The segment information for a spline move is defined in an array. An array of positions has to be defined together with optional arrays of velocities and segment durations. The arrays to use are specified with the **SPLINETABLE** keyword. Axis positions can be interpreted in one of three ways:

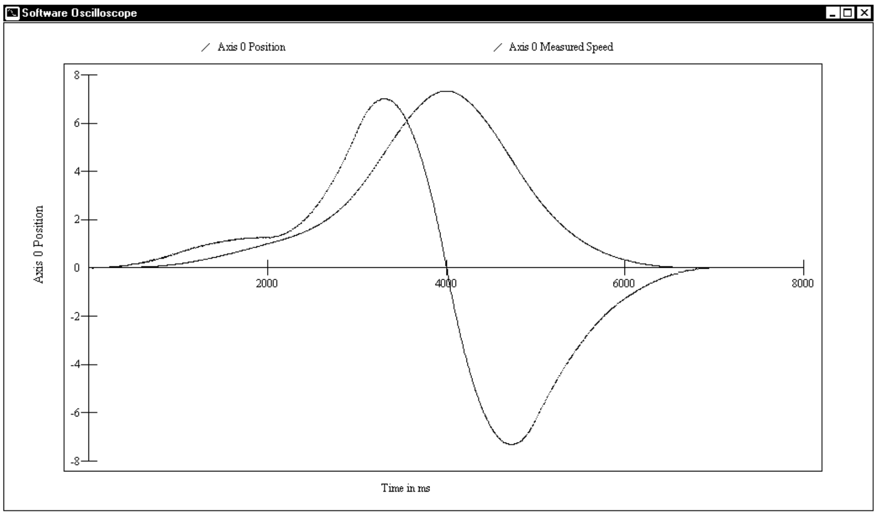
1. The values represent relative movements. Each value is the relative distance that the axis must move over that segment.
2. The values represent absolute positions within a spline cycle. During a continuous spline, each repetition of the segments is known as a cycle. Each time the cycle starts, the current axis position is considered as zero and the position values are taken as absolute to that point.
3. The values represent true absolute positions. The value are absolute with reference to actual axis position. The zero position is from when the axis was last reset or homed.

Example:

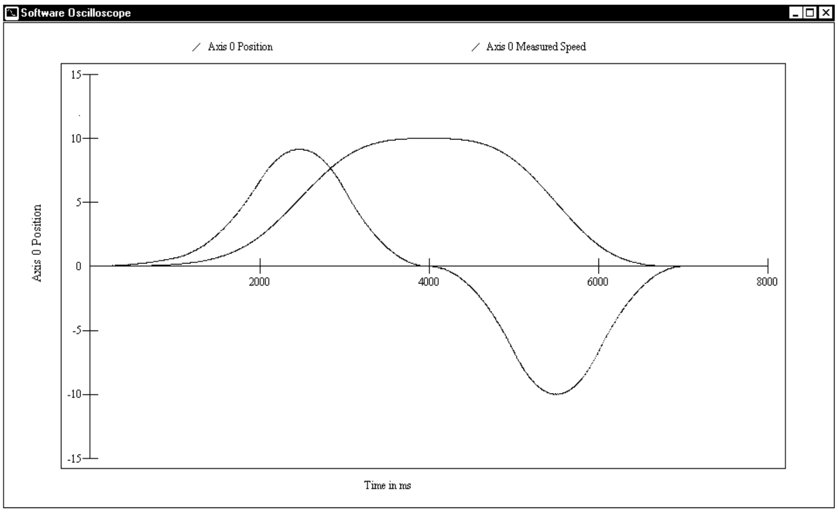
```
dim exampleArray (11) = 10, 2, 4.5, 6, 5.7, 2, 0.1, -3.4, -9.3, -6.1, 0

SPLINETABLE(0, exampleArray, null, null)
SPLINE.0 = _spABSOLUTE
```

The above defines an array of positions and uses the **SPLINETABLE** keyword to specify the name of the array to use. The value passed to the **SPLINE** keyword indicates how the data is to be interpreted. In the example, the data is to be taken as absolute positions.



When the spline is executed, each segment is processed in turn until the last segment is reached. If the spline has been specified as continuous, then the start segment will be processed at completion of the end segment. The start and end segments are set with the **SPLINESTART** and **SPLINEEND** keywords. The currently executing spline segment number can be read with **SPLINEINDEX** keyword.



The duration of each spline segment is defined with the **SPLINETIME** keyword or using the segment duration array in the **SPLINETABLE** keyword. The axis will move the distance given in the position table over the specified segment duration.

A spline can be performed on a servo or stepper axis. The **ACCEL**, **DECEL** and slew speed parameters (**SPEED** and **FEEDRATE**) are not used. Care must taken when defining the segments as the algorithm used will demand any velocity required to complete the specified segment, regardless of whether the motor can achieve that motion.

There are three types of spline motion available:

Spline 1

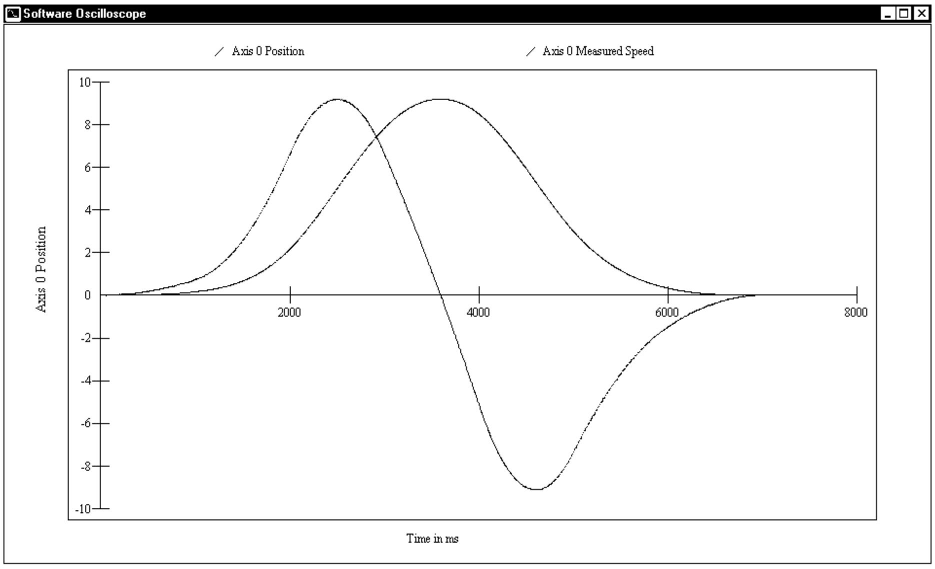
Spline 1 type motion is the default type. A position array must be defined and segment durations may be specified with the **SPLINETIME** keyword or with a duration array. The profile produced will be continuous in position, velocity, acceleration and jerk (rate of change of acceleration). Depending on the requested positions, the profile may not pass through the points specified in the position array.

Example:

```
REM Position array.  First element specifies number of position points
DIM exampleArray (6) = 5, 1, 2, 10, 2, 0

SPLINETABLE(0, exampleArray, null, null)  : REM Name arrays to use
SPLINETIME.0 = 1000                       : REM Segment duration of 1000ms
```

```
SPLINE.0 = _spSPLINE_1 + _spABSOLUTE      : REM Absolute spline 1
GO.0                                      : REM Start motion
```



The profile is shown above. The position curve shows that the axis did not achieve position 10 since a spline 1 move is not guaranteed to pass through the specified position points. The other point to note is the length of the move. A spline 1 move doubles the segment duration specified for the first and last segments of the profile. This only applies to the start and stop situation however. If the spline is continuous, then the first and last segments are of the user specified duration apart from the initial start and final stop situations. How close the profile comes to position segment point depends on the proximity of the preceding position. For example, if the position array was modified to:

```
DIM exampleArray (6) = 5, 1, 9, 10, 2, 0
```

The position profile now almost reaches position 10. The profile may be forced to reach a position by specifying the same position three times in the position array. This results in profile reaching the third repetition of the segment position with zero velocity.

```
DIM exampleArray (6) = 5, 1, 10, 10, 10, 0
```

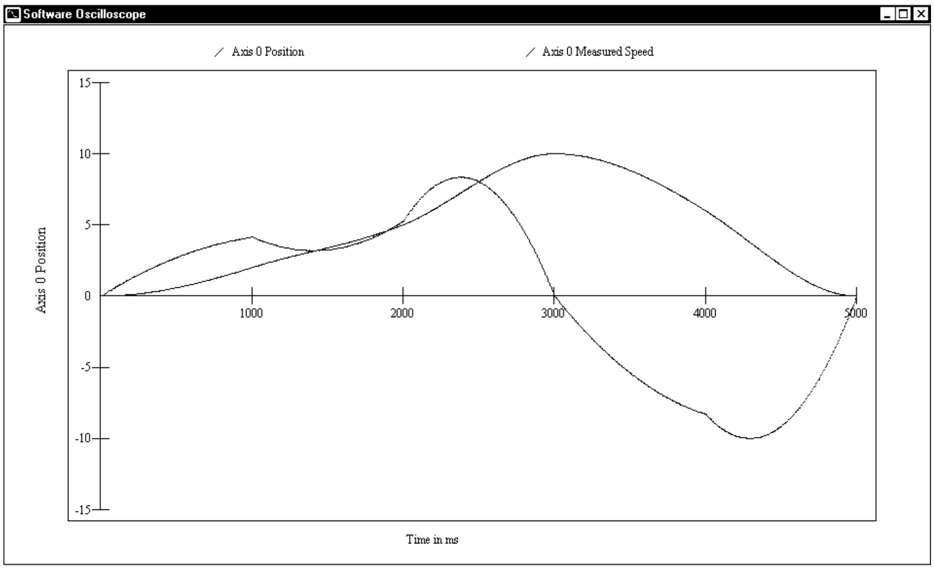
Spline 2

Spline 2 type motion produces a profile that does reach all the position points specified in the position array but the profile is only continuous in position and velocity. A position array must be defined and segment durations may be specified with the **SPLINETIME** keyword or with a duration array.

Example:

```
REM Position array.  First element specifies number of position points
DIM exampleArray (6) = 5, 2, 3, 5, -4, -6

SPLINETABLE(0, exampleArray, null, null)  : REM Name arrays to use
SPLINETIME.0 = 1000                       : REM Segment duration of 1000ms
SPLINE.0 = _spSPLINE_2                    : REM Relative spline 2
GO.0                                      : REM Start motion
```



All position points are reached and each segment takes the specified duration.

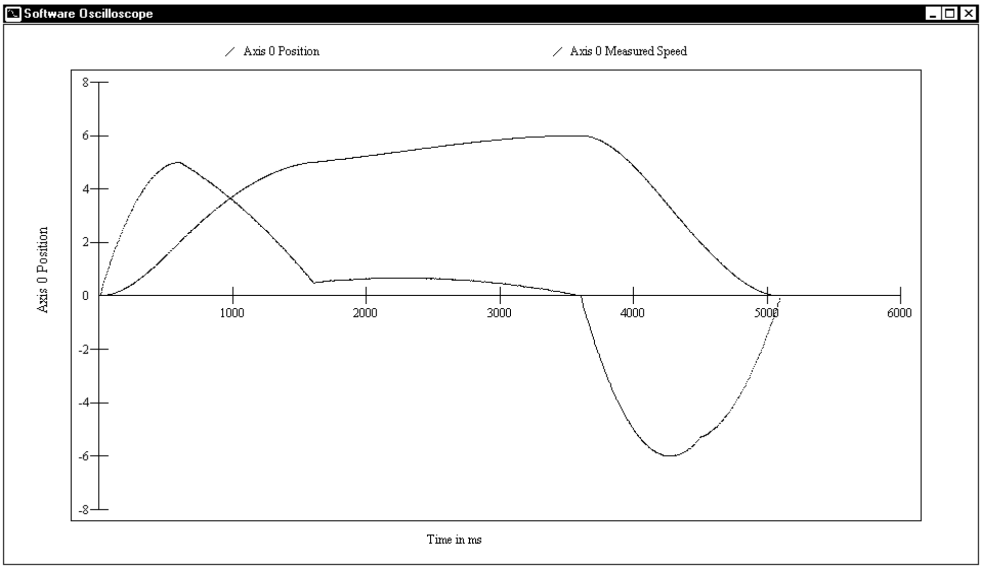
PVT

A PVT allows both position and velocity data to be specified for a profile. The profile produced reaches all the position points specified in the position array at the velocity specified in the velocity array but the profile is only continuous in position and velocity. The segment durations may be specified with the **SPLINETIME** keyword or with a duration array.

Example:

```
REM Position array.  First element specifies number of position points
DIM pArray (6) = 5, 2, 5, 6, 2, 0
REM Velocity array.  Only the number of segments required
DIM vArray (5) = 5, 0.5, 0, -5.3, 0
REM Duration array.  Only the number of segments required
DIM tArray (5) = 600, 1000, 2000, 900, 600

SPLINETABLE(0, pArray, vArray, tArray)    : REM Name arrays to use
SPLINE.0 = _spABSOLUTE                    : REM Absolute PVT
GO.0                                      : REM Start motion
```



The profile reaches each requested segment position at the specified velocity. Each segment takes the specified duration to complete

The **SPLINE** keyword sets up a spline move using the arrays specified with the **SPLINETABLE** keyword. The functionality of the spline is controlled with the mode parameter. Once started with the **GO** keyword, the spline executes in the background. It will continue until the spline comes to a natural end or the move is terminated by the user with **STOP**, **CANCEL** or **RESET** or by an error condition. The axis must be idle before calling **SPLINE** and no further moves may be loaded.

SPLINE accepts the following bit pattern:

Bit	Meaning
0	Use the positions specified in the position array as being absolute within each spline. If this bit is not set, the positions will be taken as relative movements.
1	Use the positions specified in the position array as being absolute to true motor positions. Bit 0 must be set for this to occur.
2	Continuously cycle through the spline. When the end segment is reached, the cycle starts from the start segment.
3	Perform a Spline 2 profile. If this bit is not set then the spline will be a Spline 1 profile.

If the **SPLINETABLE** keyword specifies a velocity array, then the **SPLINE** will be a PVT profile, regardless of the setting of bit 3.

There are a number of Mint constants relating to splines, these are:

Mint Constant	Description	Value
_spSPLINE_1	Perform a Spline 1 profile.	0
_spABSOLUTE	Use the positions specified in the position array as being absolute within each spline.	1
_spT_ABSOLUTE	Use the positions specified in the position array as being absolute to true motor positions.	2
_spCONTINUOUS	Continuously cycle through the spline. When the end segment is reached, the cycle starts from the start segment.	4
_spSPLINE_2	Perform a Spline 2 profile.	8

A spline move may be stopped by the **STOP** keyword, a stop switch input, an error condition or paused using a stop switch input or the **SUSPEND** keyword. In these situations, the spline will ramp to a controlled stop over a duration specified with the **SPLINESUSPENDTIME** keyword. This is particularly useful in situations where short segments are running at high velocity since the axis would not be able to decelerate in a controlled manner in that duration.

If the axis is allowed to resume from a pause situation, the axis will travel to the current segment end position over the **SPLINESUSPENDTIME** duration.

Example:

```
REM Input 3 is the stop switch input.  The axis should pause if active
STOPINPUT.4 = 3            : REM Assign input
STOPMODE.4 = 1             : REM Pause if switch active
SPLINESUSPENDTIME.4 = 2000 : REM Ramp to halt over 2000 ms.
..
REM Load spline...
```

Reading the **SPLINE** keyword will return the mode of the currently executing move. It is often desirable to load a continuous spline and then stop it at the end segment after a number of cycles. To make a continuous spline non-continuous, the **SPLINE** keyword is written to with the same value but without bit 2 set.

Example:

```
REM Stop a spline from being continuous
IF SPLINE.0 & _spCONTINUOUS DO       : REM Check is continuous
  REM Bitwise AND SPLINE with !bit 2
  SPLINE.0 = SPLINE.0 & ~_spCONTINUOUS
ENDIF
```

The current spline segment index is read using the **SPLINEINDEX** keyword. For example:

```
PAUSE SPLINEINDEX.3 = 5
OUT1 = 1
PAUSE SPLINEINDEX.3 = 9
OUT1 = 0
```

When the 5th segment is executed, output bit 1 is set. The output is cleared on the 9th segment.

See Also:

CAM, SPLINEEND, SPLINEINDEX, SPLINESTART, SPLINESUSPENDTIME, SPLINETABLE, SPLINETIME

SPLINEEND/SPE

Purpose:

To define the end segment in the SPLINE table for a SPLINE move.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
SPLINEEND[axes] = <expression> {,<expression>}
v = SPLINEEND[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●			1 ≤ x ≤ 16777215

Description:

A spline move will continue through the spline table, defined with **SPLINETABLE**, until the segment set as the end segment is reached. If the spline is continuous then the move will jump to the start segment. The end segment must be within number of segments in the currently specified spline table.

By default the number of segments in a table is given by the first element in the spline table. If multiple spline tables are required, the start and end segments can be defined.

Example:

```
DIM SplinePositions(21) = 20,10,20,30,40,50,60,70,80,90,100,
                          100,90,80,70,60,50,40,30,20,10
SPLINETABLE (0, SplinePositions, NULL, NULL)
SPLINESTART = 10
SPLINEEND = 20
SPLINE = 4
GO
```

The **SPLINESTART** keyword is used to define the start segment and **SPLINEEND** to define the end segment. Therefore, for the above example, the spline move will be based on segments 10 to 20 (values 100 to 10).

When the **SPLINE** keyword is executed, the start segment is set to 1 and the end segment set to the last segment in the table. If the start or end points are to be moved, it must be done after the **SPLINE** keyword has been executed. The start and end segments can be changed before or after the spline move has been started (**GO**).

Note that it is assumed that the first element has an array index value of 0, and the segment values start at 1.

See also:

SPLINE, SPLINEINDEX, SPLINESTART, SPLINETABLE, SPLINETIME

SPLINEINDEX/SPI

Purpose:

Returns the currently executing SPLINE segment number.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
v = SPLINEINDEX[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●						1 ≤ x ≤ 16777215

Description:

When a spline move is performed, the currently executing segment in the table can be read using the **SPLINEINDEX** keyword.

```
? SPLINEINDEX
```

This will display the current segment number. To use this value as in index into the position array then 1 must be added to the value as the first array specifies the number of segments.

```
? myPosition (SPLINEINDEX + 1)
```

This will display the position value for the currently executing spline from the array *myPosition*. In order to interpret the value as relative or absolute, the **SPLINE** keyword can be read to see the mode of the move.

If no spline move is in progress then the last segment number from the last spline move is returned.

See also:

SPLINE, SPLINEEND, SPLINESTART, SPLINETABLE, SPLINETIME

SPLINESTART/SPS

Purpose:

To define the start segment in a SPLINE table for a SPLINE move.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
SPLINESTART[axes] = <expression> {,<expression>}
v = SPLINESTART[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		1	1 ≤ x ≤ 16777215

Description:

A SPLINE move will continue through the spline table, defined with **SPLINETABLE**, until the segment set as the end segment is reached. If the spline is continuous then the move will jump to the start segment defined with **SPLINESTART**. The start segment must be within number of segments in the currently specified spline table.

By default the number of segments in a table is given by the first element in the spline table. If multiple SPLINE tables are required, the start and end segments can be defined.

Example:

```
DIM SplinePositions(21) = 20,10,20,30,40,50,60,70,80,90,100,
                          100,90,80,70,60,50,40,30,20,10
SPLINETABLE (0, SplinePositions, NULL, NULL)
SPLINESTART = 10
SPLINEEND = 20
SPLINE = 4
GO
```

The **SPLINESTART** keyword is used to define the start segment and **SPLINEEND** to define the end segment. Therefore, for the above example, the spline move will be based on segments 10 to 20 (values 100 to 10).

When the **SPLINE** keyword is executed, the start segment is set to 1 and the end segment set to the last segment in the table. If the start or end points are to be moved, it must be done after the **SPLINE** keyword has been executed. The start and end segments can be changed before or after the spline move has been started (**GO**).

Note that it is assumed that the first element has an array index value of 0, and the segment values start at 1.

See also

SPLINE, SPLINEEND, SPLINEINDEX, SPLINETABLE, SPLINETIME

SPLINESUSPENDTIME/SPD

Purpose:

To set the segment duration for a controlled stop during a spline move.

Controller Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
SPLINESUSPENDTIME[axes] = <expression> {,<expression> ...}
v = SPLINESUSPENDTIME[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		1000	4 to 60000

Description:

Each spline is executed over a specified time period. For normal operation, the duration is set with an array of durations or with the **SPLINETIME** keyword.

In the event of a controlled stop, e.g. the **STOP** keyword is issued or an asynchronous error mode indicates a controlled stop, then a spline move will ramp to zero speed over the number of milliseconds specified with the **SPLINESUSPENDTIME** keyword.

When allowed to resume motion, the axis will continue to the current end of segment position with a duration of **SPLINESUSPENDTIME**.

Example:

```
SPLINETABLE (0, myArray, NULL, NULL) : REM Define a position array
SPLINETIME.0 = 100                   : REM Each segment lasts 100ms
SPLINESUSPENDTIME.0 = 2000           : REM Ramp to halt over 2000ms
SPLINE.0 = 0                         : REM Relative spline 1
GO.0
```

The minimum spline segment duration is 4 milliseconds, with a minimum resolution of 2ms. The maximum is 60000ms.

See also:

SPLINE, SPLINEEND, SPLINEINDEX, SPLINESTART, SPLINETABLE, SPLINETIME

SPLINETABLE()

Purpose:

To specify the array names to be used in a SPLINE move on the specified axis.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
SPLINETABLE ( axis, position array, {velocity array}, {duration array} )
```

Dot Parameters:

Axis – Axis No
Position Array – Array of positional data
Velocity Array – Array of velocity data
Duration Array – Array of segment duration data

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove			●				

Description:

A spline move is defined by splitting the motion into segments which form the spline table. The information about these segments are stored in normal user arrays which can have any name. Three tables can be defined for a spline move, the position table, velocity table and duration table.

Each segment of the position array specifies how far and in which direction the slave axis will move for that segment. The table is defined using an array which defines the number of segments in the table and the table itself. The first element of the array specifies the number of segments defined in the table and the subsequent values define the SPLINE segments. The values are in user units, as set with the **SCALE** keyword and the table can have any number of segments, (memory space permitting).

When the spline is executed, each segment is processed in turn until the end of the table is reached. A single shot spline will stop when the end of the table is reached. A continuous spline will repeat the table when the end is reached. The **SPLINE** keyword controls whether the spline is single shot or continuous. The positions in the table can be interpreted in three ways; relative, absolute to the cycle or true absolute. See the **SPLINE** keyword for full details.

The velocity table allows for end of segment velocities to be defined. The velocities are defined in user units/second as set with the **SCALE** keyword. If velocity information is defined, then the spline motion will only not be continuous in acceleration or jerk. If the no velocity array is required for a spline move, then **NULL** must be passed instead.

Each spline segment is executed over a specified time duration. All segments can be set to use the same duration with the **SPLINETIME** keyword. Alternatively, a table of segment durations can be defined. The durations are in seconds and are relative, not absolute to the start of motion.

The ability to define a duration table allows for critical points on a move to be defined more easily. Where the motion is not critical, a large durations can be set so the axis travels a large distance in only one or two segments. Then where the motion is more important, smaller durations are defined, allowing the segment moves to be smaller and more detailed. The minimum segment duration is 8 milliseconds. If the duration table is not required for a SPLINE move then **NULL** must be passed instead.

The minimum spline segment duration is 4 milliseconds, with a minimum resolution of 2ms.

The **SPLINETABLE** keyword is used to specify which arrays to use for a, spline move on the axis. The velocity and duration array names are optional. The arrays will be used on all further spline moves on the axis until the next call to **SPLINETABLE**.

Example:

To set up a position table called *myPositions*, a velocity table called *myVelocities* and a duration table called *myDurations* and use these for a SPLINE move on axis 1:

```
DIM myPositions (21) = 20, 1, 2, ...
DIM myVelocities (20) = 1, 2, ...
DIM myDurations (20) = 1000, 2000, ...
SPLINETABLE ( 1, myPositions, myVelocities, myDurations )
```

To just use the position and velocity tables:

```
SPLINETABLE ( 1, myPositions, myVelocities, NULL )
```

See also:

CAMTABLE, SPLINE, SPLINEEND, SPLINEINDEX, SPLINESTART, SPLINETIME

SPLINETIME/SPT

Purpose:

To set the segment duration for all segments for a SPLINE move.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
SPLINETIME[axes] = <expression> {,<expression> ...}
v = SPLINETIME[axis]
```

Dot Parameters:

Axis - Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●	●	1.0	$0.0 \le x \le 8388607.0$

Description:

Each spline segment is executed over a specified time period. Either a global time period can be set for all segments or segments can have their durations individually set.

The **SPLINETIME** keyword is used to set the global segment duration, the value being specified in milliseconds. The duration can be changed while the spline move is in progress but it will not take effect until the following segment has been completed.

Example:

```
SPLINETABLE (1, myArray, myVel, NULL)
SPLINETIME.1 = 30                   : REM Duration of 30ms per segment
SPLINE.1 = _spABSOLUTE
GO.1
```

This sets up and starts a spline move on axes 0 and 1. The segment duration is set to 50 milliseconds.

The minimum spline segment duration is 4 milliseconds, with a minimum resolution of 2ms. The maximum is 60000ms.

See also:

SPLINE, SPLINEEND, SPLINEINDEX, SPLINESTART, SPLINETABLE

SRAMP/SRP

Purpose:

To set the smoothness of the velocity profile.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Dot Parameters:

Axis - Axis No.

Format

```
SRAMP[axes] = <expression> {,<expression> ...}
v = SRAMP[axis]
```

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	$0 \le x \le 100.0$

Description:

It is possible to impose a form of modified S-ramping to the standard trapezoidal move profile using the **SRAMP** keyword. The velocity profile of positional moves is trapezoidal, specified by the acceleration, **ACCEL**, deceleration, **DECEL**, slew speed, **SPEED** or **FEEDRATE**, and move distance. The normally linear acceleration and deceleration ramps may be smoothed to approximate 'S' ramps. The degree of smoothing is controlled by **SRAMP** value where a high value produces a very rounded profile while a value of zero results in a true trapezoid.

SRAMP affects the following move types:

- **CIRCLEA**, **CIRCLER**
- **MOVEA**, **MOVER**
- **VECTORA**, **VECTORR**

In multi-axis moves the ramp factor on the master axis will be applied to all axes involved in the move.

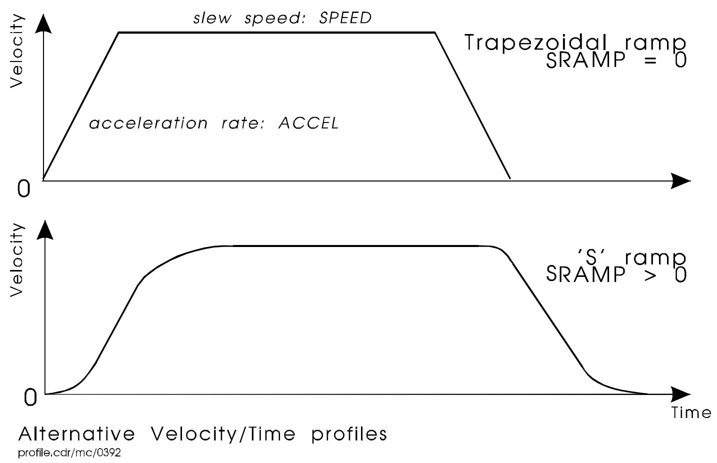


Figure 6: Effect of SRAMP on a move

Example:

```
SRAMP = 5,5
ACCEL = 500,500
SPEED = 20,20

CIRCLER = 100,200,45 : GO
```

The **SRAMP** factor cannot be changed while a move is in progress.

See also:

ACCEL, CIRCLEA, CIRCLER, DECEL, FEEDRATE, MOVEA, MOVER, SPEED, VECTORA, VECTORR

STEPPERIO/SI

Purpose:

Manually control the pulse and direction pins of a stepper channel.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●			

Format:

```
STEPPERIO.channel  = <expression>
v = STEPPERIO.channel
```

Dot Parameters:

Channel – Stepper channel number.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	$0 \le x \le 3$

Description:

Control the pulse and direction pins of a stepper axis.

Bit	Action
0	Controls the PULSE pin
1	Controls the STEP DIRECTION pin

Example:

```
CONFIG.4 = _cfOFF
STEPPERIO.4 = 0x3
```

This turns on both the pulse and the step direction pin of axis 4.

See Also:

BOOST, FREQ

SUSPEND/SSD

Purpose:

To pause the current move.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Dot Parameters:

Axis - Axis No.

Format:

```
SUSPEND[axes] = <expression> {,<expression> ...}
v = SUSPEND[axis]
```

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●		●		0	$0 \le x \le 1$

Description:

The **SUSPEND** keyword can be used to temporarily halt a move. The move will ramp down to zero speed and remain stationary until the move is resumed. The mode of motion will continue to show the move type whilst the move has been suspended.

SUSPEND takes a value of 0 or 1:

1 : Suspend a move
0: Resume the move.

SUSPEND only applies to the following move types:

- **MOVEA**/**MOVER**
- **VECTORA**/**VECTORR** (NextMove only)
- **CIRCLEA**/**CIRCLER** (NextMove only)
- **HOME** (NextMove only)
- **JOG**
- **INCA/INCR** (MintDrive & ServoNode 51 only)
- **FOLLOW** (**FOLLOWMODE** 1 - NextMove only)

Example:

```
LOOP
  key = INKEY
  IF key = ‘S’ THEN SUSPEND = 1;  : REM Suspend motion
  IF key = ‘R’ THEN SUSPEND = 0;  : REM Resume motion
ENDL
```

In this example, the program loops checking for key presses with **INKEY**. If the letters S or R have been pressed and all axes are either brought to a halt or restarted.

See Also:

#STOP, STOP, STOPSWITCHMODE

TERMINALMODE/TMD

Purpose:

To set the overwrite mode of the terminal input/output channels.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
TERMINALMODE.channel = <expression>
v = TERMINALMODE.channel
```

Dot Parameters:

Channel – serial channel.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●	●				0	0 or 1

Description:

Many of the terminal channels, such as DPR or RS232, expect a host computer at the other end to collect the data. If there is no host computer, then the Mint program can lock up waiting for the serial characters to be read. This is due to the handshaking between the Mint controller and host such that characters are not lost.

TERMINALMODE is used to turn off handshaking, i.e. when the serial buffer is full, the contents are lost.

Bit	Constant	Description
0	_tmmOVERWRITE	Turns on transmit overwrite. If the transmit buffer gets full, characters will be lost.
1	_tmmOFF	Turns the transmit channel off. All characters written to this channel will be lost. Pressing Ctrl-E on this channel will restore the channel. Setting this bit overrides the **TERMINAL** setting.

Example:

```
TERMINALMODE._tmRS232 = 010
```

See Also:

CLS, ECHO, PRINT, TERMINAL

VELDEMAND/VLD

Purpose:

Read the current instantaneous demand velocity.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●	●	●

Format:

```
v = VELDEMAND[axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
All	●			●	●		

Description:

Reads the current instantaneous demand velocity as generated by the profiler.

Example:

```
dim currentVel=0
currentVel = VELDEMAND.0
```

This get the current demand velocity for axis zero

See Also:

POSDEMAND

VELENCODER/VEN

Purpose:

To select the source of the velocity signal used in dual encoder feedback systems.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
VELENCODER.axis = <expression>
value = VELENCODER.axis
```

Dot Parameters:

axis – the axis number.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●			0 – 3 (0 – 7 on PCI with expansion card)

Description:

Dual encoder feedback is a control technique where two encoders are used to control a single axis. Generally one of the encoders is attached to the load and the other to the motor. The encoder attached to the load gives high positional accuracy and the encoder attached to the motor provides stability.

The implementation of *Dual Encoder Feedback* uses the two encoders to close two control loops, position and velocity. The encoder attached to the load is used to close the positional loop and the encoder attached to the motor shaft is used to close the velocity loop. This enables high positional accuracy to be achieved via the encoder attached to the load, yet the stability problems associated with coupling imperfections (compliance and backlash) are eliminated as these are now inside the velocity loop.

The **VELENCODER** keyword is used to specify the *encoder channel* that the axis will use for velocity feedback. The encoder can be changed at any time though it is recommended that the axis is disabled when this is done. The velocity used in the servo to be multiplied by the **KVEL** term comes from the velocity encoder. The **VEL** keyword and 'measured speed' in data capture are from the velocity encoder. Positional information is read from the standard axis encoder.

The velocity encoder is specified as a channel number in the range 0 – 3. On NextMove PCI, if an expansion card is present, then range increases to 0-7. Auxiliary encoder channels may also be specified as shown in the following table:

Auxiliary encoder channel	Value to write to VELENCODER
0 (NextMove BX and NextMove PCI)	-1
1 (NextMove PCI with expansion card)	-2

To turn off dual encoder feedback, the velocity encoder must be set to the same encoder as is being used for position feedback. This can be achieved by reading the **AXISCHANNEL** keyword.

For example, turn off dual encoder feedback on axis 0.

```
VELENCODER.0 = AXISCHANNEL.0
```

Example 1

On NextMove PCI, axis 2 controls a table through a leadscrew. There are two encoders available, one attached to the table and another to the motor shaft. Axes 0, 1 and 3 are also in use. To set this system up for dual encoder feedback, the position feedback encoder (on the table) should be attached to encoder input 2. The velocity encoder (on the motor) should be attached to the auxiliary encoder input. The **VELENCODER** keyword is used to specify the location of the velocity feedback.

```
VELENCODER.2 = -1
```

Example 2

On NextMove BX, axis 0 provides position feedback and axis 1 provides velocity feedback. The **VELENCODER** keyword used to specify that axis 0 is using dual encoder feedback.

```
VELENCODER.0 = 1
```

See also:

AXISCHANNEL, CONFIG

VELFATAL/VLF

Purpose:

Set a threshold for the maximum difference between demand and actual velocity.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
VELFATAL [axes] = <expression> {,<expression> ...}
v = VELFATAL[axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●	●	8388607.0	$0.0 < x \leq 8388607.0$

Description:

Velocity error checking allows the measured velocity of an axis to be compared to its demand velocity. If the difference between the two values exceeds the limit set with **VELFATAL**, then an error will be created. This allows for a burst of noise or poor drive performance to be detected.

If the velocity error exceeds the fatal value, an asynchronous error will be generated and bit 17 of **AXISERROR** will be set. The response to this error is controlled with the function **VELFATALMODE**.

See Also:

AXISERROR, VELFATALMODE

VELFATALMODE/VLM

Purpose:

Specify the action taken in the event of the velocity threshold being exceeded.

Controllers Supported:

NextMove PCI	NextMove PC	NextMove BX	MintDrive	ServoNode 51
●	●	●		

Format:

```
VELFATALMODE[axes] = <expression> {,<expression> ...}
v = VELFATALMODE[axis]
```

Dot Parameters:

Axis – Axis No.

Attributes:

Controller	Read	Write	Command	Multi-Axis	Scaled	Default	Range
NextMove	●	●		●		0	$0 \le x \le 6$

Description:

The function **VELFATALMODE** controls what action is taken in the event of the **VELFATAL** limit being exceeded.

The possible modes are as follows.

Mode	Action
0	Ignore the error condition.
1	Crash stop the axis and drop the enable line. When the axis is idle[8] the error handler will be called.
2	Crash stop the axis, leave the axis enabled. When the axis is idle the error handler will be called.
3	Perform a controlled stop on the axis at the rate specified by the **ERRORDECEL** parameter, leave the axis enabled. When the axis is idle the error handler will be called.
4	Not applicable
5	Call the error handler only.
6	Ramp the DAC to zero with a rate set with **DACRAMP**. The axis will be disabled when the DAC reaches zero and the error handler called.

See Also:

VELFATAL

[8] An axis is deemed to be idle if it is disabled or if it has a following error.

NextMove Mappings

7

This section contains the default axis mapping information for NextMove controllers.

7.1 NextMove PCI

NextMove PCI expansion cards are sold in two variants; 4 axes of control and 8 axes of control. The number of axes of control on an expansion card adds to the number of axes of control on the main card, subject to a maximum of 12 axes of control.
To check that the expansion card has been detected, enter **VIEW HARDWARE** at the Mint command prompt. After the controller production parameters have been displayed, if an expansion card has been detected, this will be indicated:

```
  ...
  Firmware Downloads          : 3

NextMove PCI axis expansion card 1 detected.
```

With an expansion card attached, the default axis configurations and hardware used may change. Use the **VIEW CONFIG** keyword to see the default assignments.

The **NUMBEROF** keyword can also be used to return information of available axes and channels. The **CHANNELTYPE** keyword can be used to determine the hardware types available for a given channel number.

7.1.1 Open Loop Access

The hardware channels on main and expansion cards have a fixed numbering system. Any open loop access to hardware will always be made through the channel number. The following tables show how the open loop channel number corresponds to the labeling on the breakout unit.

Main Card – M1, M2, M3, M4, M8			
Open loop channel	**DAC**	**Encoder**	**Pulse Direction**
0	0	A	E
1	1	B	F
2	2	C	G
3	3	D	H

Expansion Card 1 - E4, E8			
Open loop channel	**DAC**	**Encoder**	**Pulse Direction**
4	0	A	E
5	1	B	F
6	2	C	G
7	3	D	H

Expansion Card 2 - E4, E8			
Open loop channel	**DAC**	**Encoder**	**Pulse Direction**
8	0	A	E
9	1	B	F
10	2	C	G
11	3	D	H

where:

M1 - NextMove PCI with 1 axis of control
M2 - NextMove PCI with 2 axes of control
M3 - NextMove PCI with 3 axes of control
M4 - NextMove PCI with 4 axes of control
M8 - NextMove PCI with 8 axes of control
E4 - NextMove PCI expansion with 4 axes of control
E8 - NextMove PCI expansion with 8 axes of control

For example, to write a value to the first DAC on the main card in open loop mode, the DAC keyword would be called with a channel number of 0. The set the pulse or direction pins of the first stepper outputs in open loop mode would also use a channel number of 0.

Open loop access is available as long as no axis mapped to the channel configured to be using the hardware being accessed in open loop mode. For example, if axis 0 is mapped to channel 0 and the axis is configured as a servo, then the pulse / direction outputs of channel 0 can be used.

A main card has 8 physical resources (4 channels, servo and stepper hardware per channel). Where a M1, M2 or M3 variants is purchased, the remaining hardware channels will NOT be available in either open or closed loop modes.

7.1.2 Default Mappings

On power up, a set of default mappings and configurations will be made.

M1, M2, M3, M4		
Axis Number	**Channel**	**Config**
0	0	servo
1	1	servo[9]
2	2	servo[9]
3	3	servo[9]
4 - 11	-1	off

(M1, M2, M3, M4) + E4		
Axis Number	**Channel**	**Config**
0	0	servo
1	1	servo[10]
2	2	servo[10]
3	3	servo[10]
4	4	servo
5	5	servo
6	6	servo
7	7	servo
8 – 11	-1	off

(M1, M2, M3, M4) + (E4 + E4) / (E8 + E4) / (E8 + E8)		
Axis Number	**Channel**	**Config**
0	0	servo
1	1	servo[10]
2	2	servo[10]
3	3	servo[10]

[9] If number of axes of control allow
[10] If number of axes of control allow

(M1, M2, M3, M4) + (E4 + E4) / (E8 + E4) / (E8 + E8)		
Axis Number	**Channel**	**Config**
4	4	servo
5	5	servo
6	6	servo
7	7	servo
8	8	servo
9	9	servo
10	10	servo
11	11	servo

M8		
Axis Number	**Channel**	**Config**
0	0	servo
1	1	servo
2	2	servo
3	3	servo
4	0	stepper
5	1	stepper
6	2	stepper
7	3	stepper
8 - 11	-1	off

M8 + E4, M8 + E8, M8 + (E4 + E4) / (E4 + E8) / (E8 + E8)		
Axis Number	**Channel**	**Config**
0	0	servo
1	1	servo
2	2	servo
3	3	servo
4	0	stepper
5	1	stepper
6	2	stepper
7	3	stepper
8	4	servo
9	5	servo
10	6	servo
11	7	servo

7.2 NextMove PC

The NextMove PC axis expansion card is sold in two variants; 4 axes of servo and stepper control and 4 axes of stepper control. The total number of axes of control will not exceed 8 of NextMove PC.
To check that the expansion card has been detected, enter **`VIEW HARDWARE`** at the Mint command prompt. After the controller production parameters have been displayed, if an expansion card has been detected, this will be indicated:

With an expansion card attached, the default axis configurations and hardware used may change. Use the **`VIEW CONFIG`** keyword to see the default assignments.

The **`NUMBEROF`** keyword can also be used to return information of available axes and channels. The **`CHANNELTYPE`** keyword can be used to determine the hardware types available for a given channel number.

7.2.1 Open Loop Access

The hardware channels on main and expansion cards have a fixed numbering system. Any open loop access to hardware will always be made through the channel number. The following tables show how the open loop channel number corresponds to the labeling on the breakout unit.

Main Card – M1, M2, M3, M4, M8			
Open loop channel	**DAC**	**Encoder**	**Pulse Direction**
0	0	A	E
1	1	B	F
2	2	C	G
3	3	D	H

Expansion Card - ESS, ES			
Open loop channel	**DAC**	**Encoder**	**Pulse Direction**
4	0	A	E
5	1	B	F
6	2	C	G
7	3	D	H

where:

M2 - NextMove PC with 2 axes of control
M3 - NextMove PC with 3 axes of control
M4 - NextMove PC with 4 axes of control
M8 - NextMove PC with 8 axes of control
ESS - NextMove PC expansion with 4 servo and 4 stepper axes
ES - NextMove PC expansion with 4 stepper axes

For example, to write a value to the first DAC on the main card in open loop mode, the DAC keyword would be called with a channel number of 0. The set the pulse or direction pins of the first stepper outputs in open loop mode would also use a channel number of 0.

Open loop access is available as long as no axis mapped to the channel configured to be using the hardware being accessed in open loop mode. For example, if axis 0 is mapped to channel 0 and the axis is configured as a servo, then the pulse / direction outputs of channel 0 can be used.

7.3 NextMove BX

NextMove BX is sold in 2, 3 and 4 axis variants. Use the **`VIEW CONFIG`** keyword to see the default assignments.
The **`NUMBEROF`** keyword can also be used to return information of available axes and channels. The **`CHANNELTYPE`** keyword can be used to determine the hardware types available for a given channel number.

7.3.1 Open Loop Access

The hardware channels on main card have a fixed numbering system. Any open loop access to hardware will always be made through the channel number. The following tables show how the open loop channel number corresponds to the labeling on the breakout unit.

Main Card – M1, M2, M3, M4, M8		
Open loop channel	**DAC**	**Encoder**
0	0	A
1	1	B
2	2	C
3	3	D

where:

M2 - NextMove BX with 2 axes of control
M3 - NextMove BX with 3 axes of control
M4 - NextMove BX with 4 axes of control

For example, to write a value to the first DAC on the main card in open loop mode, the DAC keyword would be called with a channel number of 0. Open loop access is available as long as no axis mapped to the channel configured to be using the hardware being accessed in open loop mode.

7.3.2 Default Mappings

On power up, a set of default mappings and configurations will be made.

M1, M2, M3, M4		
Axis Number	**Channel**	**Config**
0	0	servo
1	1	servo
2	2	servo
3	3	servo
4 - 7	-1	off

Bibliography

8

Bibliography

[1] Mint v4 Programming Guide (MN1262)
[2] Mint v4 CAN Programming Guide (MN1282)
[3] Mint v4 PC Programming Guide (MN1278)
[4] Mint v4 Embedded Programming Guide (MN1279)
[5] Mint v4 Function Reference Guide (MN1280)
[6] Mint v4 Code Analyzer Tool Guide (MN1281)
[7] Mint WorkBench Users Guide (MN1283)
[8] NextMove PCI Installation Guide (MN1277)
[9] NextMove PC Installation Guide (MN1257)
[10] NextMove BX Installation Guide (MN1258)
[12] CAN Peripherals Hardware Guide (MN1255)

All manuals can be found in the Baldor Motion Toolkit CD which accompanies the controllers.